**DRBD中文应用指南**

写在前面的话：

为了方便学习，将drbd的英文文档翻译出来，以供学习和参考。但因为本人英语水平有限，以及对drbd的了解并不足够深入，翻译中出现很多不通顺或者是语病的地方，请大家谅解。[如果问题可以反馈给15038051897@163.com](mailto:%E5%A6%82%E6%9E%9C%E9%97%AE%E9%A2%98%E5%8F%AF%E4%BB%A5%E5%8F%8D%E9%A6%88%E7%BB%9915038051897@163.com)，鄙人将在第一时间进行更正。

如果是学习外语文刊的翻译，那么他的水平也仅仅是停留在翻译者的水平，因此建议有实力的同学还是从官网获取第一手信息。

可在<http://download.csdn.net/source/3540203>中下载相关PDF，在这里很多的图片无法显示，请谅解！

 刘运锋

2011-08-22

**一．关于本指南**

本指南主要为分布式复制块设备（DRBD技术）的用户提供一个简明的参考指南和手册。

DRBD是由LINBIT项目所在的公司赞助的，免费，并比较有益处的DRBD技术社区。该指南在不断的更新，以保持和（作者）新添加在DRBD技术新功能的信息的同步。在线的HTML本指南版本可以参见<http://www.drbd.org/users-guide/>。

|  |
| --- |
| **重要**  **本指南可能有些段落包含“草稿”字样，是因为最近这些段落已增加了可能不是很权威的文档说明。欢迎广大读者反馈意见，以示对我们的鼓励。邮箱地址为：drbd-user@lists.linbit.com。** |

本指南分五个部分和一个附录：

第一部分，介绍“DRBD”：介绍DRBD技术的基本功能。简要概述DRBD技术在Linux I / O堆栈的DRBD技术的定位和基本DRBD技术概念另外还增加了被认为是DRBD技术的最重要特点。

第二部分“DRBD技术编译，安装和配置”在源代码中如何建立DRBD技术，DRBD技术编译安装包，DRBD技术的集群系统，以描述工作流程。

第三部分“DRBD技术”用在如何管理DRBD技术，DRBD技术的源码配置，修改，并介绍如何解决常见问题。

第四部分“DRBD技术应用相结合”，以增加存储的复制如何利用的DRBD技术优势和应用程序可用性。存活状态的探测，以及在与群集管理器，LVM和先进的组合的组合，整合DRBD LVM和GFS以实现高可用性,并添加到Xen虚拟化环境中。

第五部分“优化DRBD性能”：DRBD的配置获得最佳性能指南。

第六部分，“关于DRBD技术更多学习”：DRBD的内部原理的深入理解，还包含一些其他资源。希望这本指南对读者有用。

最后，附录A，DRBD技术系统手册页包含了Linux手册分发给了最新版本的DRBD技术参考之用，网页的在线版本。

对DRBD技术培训或支援服务感兴趣的用户请联系<sales@linbit.com>。

**第一篇       DRBD介绍**

目录

[1.DRBD的基本功能](http://write.blog.csdn.net/postedit/6460814#_1.DRBD的基本功能)

[1.1内核模块](http://write.blog.csdn.net/postedit/6460814#_1.1内核模块)

[1.2用户空间管理工具](http://write.blog.csdn.net/postedit/6460814#_1.2用户空间管理工具)

[1.3资源](http://write.blog.csdn.net/postedit/6460814#_1.3资源)

[1.4资源角色](http://write.blog.csdn.net/postedit/6460814#_1.4资源角色)

[2.DRBD的特点](http://write.blog.csdn.net/postedit/6460814#_2.DRBD的特点)

[2.1单主模式](http://write.blog.csdn.net/postedit/6460814#_2.1单主模式)

[2.2双主模式](http://write.blog.csdn.net/postedit/6460814#_2.2双主模式)

[2.3复制模式](http://write.blog.csdn.net/postedit/6460814#_2.3复制模式)

[2.4多复制传输](http://write.blog.csdn.net/postedit/6460814#_2.4多复制传输)

[2.5高效同步](http://write.blog.csdn.net/postedit/6460814#_2.5高效同步)

[2.6在线设备验证](http://write.blog.csdn.net/postedit/6460814#_2.6在线设备验证)

[2.7复制传输完整性验证](http://write.blog.csdn.net/postedit/6460814#_2.7复制传输完整性验证)

[2.8裂脑通知和自动修复](http://write.blog.csdn.net/postedit/6460814#_2.8裂脑通知和自动修复)

[2.9对磁盘缓冲的支持](http://write.blog.csdn.net/postedit/6460814#_2.9对磁盘缓冲的支持)

[2.10磁盘错误处理策略](http://write.blog.csdn.net/postedit/6460814#_2.10磁盘错误处理策略)

[2.11过期数据处理策略](http://write.blog.csdn.net/postedit/6460814#_2.11过期数据处理策略)

[2.12三路复制](http://write.blog.csdn.net/postedit/6460814#_2.12三路复制)

[2.13使用DRBD Proxy实现远距离复制](http://write.blog.csdn.net/postedit/6460814#_2.13使用DRBD_Proxy实现远距离复制)

[2.14基于复制的传输](http://write.blog.csdn.net/postedit/6460814#_2.14基于复制的传输)

[2.15浮动的对等节点](http://write.blog.csdn.net/postedit/6460814#_2.15浮动的对等节点)

**1.DRBD的基本功能**

分布式复制块设备（DRBD技术）是一种基于软件的，无共享，复制的存储解决方案，在服务器之间的对块设备（硬盘，分区，逻辑卷等）进行镜像。

DRBD镜像数据

**实时性：**当应用对磁盘的数据进行修改时，复制立即发生。

**透明性：**应用程序的数据存储在镜像设备上是独立和透明的，数据可存储在不同的服务器上。

**同步镜像和异步镜像：**同步镜像，当本地发申请进行写操作进行时，同步写到两台服务器上。异步镜像，当本地写申请已经完成对本地的写操作时，开始对对应的服务器进行写操作。

**1.1内核模块**

DRBD技术的核心功能是通过一个Linux内核模块实现的。具体来说，DRBD包含一个虚拟的块设备，因此DRBD是位于“右底部附近的“一个系统的I/ O堆栈。正因为如此，DRBD极为灵活，这使得它成为几乎适合任何程序的一个高可用的块复制解决方案。

|  |
| --- |
| **重要**  **DRBD技术，顾名思义，由Linux内核架构所支撑，它建立在不可知的层上面。因此，DRBD不可能添加了上层所具备的一些新特性。例如，DRBD技术不能自动检测文件系统损坏或添加双主动群集能力，像Ext3或者XFS文件系统。** |

**图1.1     DRBD在Linux的I/O堆栈中的位置**

**1.2用户空间管理工具**

为了能够管理和配置DRBD的资源，DRBD配备了一些管理工具与内核模块进行通信。

**drbdadm：**高层的DRBD程序管理套件工具。它从配置文件/etc/drbd.conf中获取所有配置参数。drbdadm为drbdsetup和drbdeta两个命令充当程序的前端应用，执行drbdadm实际是执行的drbdsetup和drbdeta两个命令。

**drbdsetup：**drbdsetup可以让用户配置已经加载在内核中运行的DRBD模块，它是底层的DRBD程序管理套件工具。使用该命令时，所有的配置参数都需要直接在命令行中定义，虽然命令和灵活，但是大大的降低了命令的简单易用性，因此很多的用户很少使用debdsetup。

**drbdmeta：**drbdmeta允许用户创建、转储、还原和修改drbd的原数据结构。这个命令也是用户极少用到。

**1.3资源**

在DRBD中，资源是所有可复制移动存储设备的总称。这些措施包括：

**资源名：**资源名可以指定除了空格外us-ascii中的任意字符。

**DRBD设备：**DRBD的虚拟块设备。它有一个主设备号为147的设备，默认的它的次要号码编从0开始。相关的块设备需命名为/ dev/ drbdm，其中M是设备的次要号码。

|  |
| --- |
| **备忘**  早期的DRBD版本中，被“劫持”的NBD设备主号码为43。这个很早就过时了，147是LANANA-注册的DRBD设备主号码。 |

**磁盘配置：**DRBD内部应用需要本地数据副本，元数据。

**网络配置：**各个对等接点间需要进行数据通信。

**1.4资源角色**

在DRBD中，每个节点都有自己的角色，比如主或者备。

|  |
| --- |
| **备忘**  这里属于的选择不是随意的，这些角色的没有被DRBD的作者命名为“主动”和“被动”。主和备的概念是依赖与存储的可用性的，而主动和被动反应的是一个应用的可用性。通常情况下在高可用环境中主节点也往往是活动的节点，但是这不是必须的。 |

在主DRBD设备中可以进行不受限制的读和写的操作。他可用来创建和挂载文件系统、初始化或者是直接I/O的快设备，等等。

在次DRBD设备中，接收所有来自对等节点的更新，但是与此同时也就完全拒绝了访问。它即不能被应用也不能被读写访问。为次节不能被读写访问是为了保持缓冲一致性，这就意味着次节点是不可能以任何形式被访问的。

人工干预和管理程序的自动聚类算法都可以改变资源的角色。资源从次节点变为主节点为升级，而反操作则成为降级。

**2.DRBD的特点**

这一章主要介绍DRBD的各种有用的特性，并介绍关于这些特性一些背景信息。其中的一部分会对大多数的用户很有用，其他的就和具体的部署方式有关系。

,普通的行政任务与第7章、故障排除和差错恢复含说明如何实现和使用这些功能,在日常操作。

将在第6章的常见的管理任务和第7章故障排除和错误恢复，介绍在日常操作管理中如何启用和利用这些特性。

**2.1单主模式**

在单主模式下，任何资源在任何特定的时间，集群中只存在一个主节点。正是因为这样在集群中只能有一个节点可以随时操作数据，这种模式可用在任何的文件系统上（EXT3、EXT4、XFS等等）。

部署DRBD单主节点模式可保证集群的高可用性（fail-over遇故障转移的能力）。

**2.2双主模式**

这是DRBD8.0之后的新特性。

在双主模式下，任何资源在任何特定的时间，集群中都存在两个主节点。由于双方数据存在并发的可能性，这种模式需要一个共享的集群文件系统，利用分布式的锁机制进行管理，如GFS和OCFS2。

部署双主模式时，DRBD是负载均衡的集群，这就需要从两个并发的主节点中选取一个首选的访问数据。这种模式默认是禁用的，如果要是用的话必须在配置文件中进行声明。

可参见“启用双主模式”一节，了解有关启用双主模式的信息。

**2.3复制模式**

DRBD提供了三种不同的复制方式，允许三种度的复制：

**协议A：**一旦本地磁盘写入已经完成，数据包已在发送队列中，则写被认为是完成的 。在一个节点发生故障时，可能发生数据丢失，因为被写入到远程节点上的数据可能仍在发送队列。尽管，在故障转移节点上的数据是一致的，但没有及时更新。这通常是用于地理上分开的节点。

**协议B：**一旦本地磁盘写入已完成且复制数据包达到了对等节点则认为写在主节点上被认为是完成的。数据丢失可能发生在参加的两个节点同时故障的情况下，因为在飞行中的数据可能不会被提交到磁盘。

**协议C：**只有在本地和远程节点的磁盘已经确认了写操作完成，写才被认为完成。没有任何数据丢失，所以这是一个群集节点的流行模式，但I/O吞吐量依赖于网络带宽。

简言之：

A 数据一旦写入磁盘并发送到网络中就认为完成了写入操作。

B 收到接收确认就认为完成了写入操作。

C 收到写入确认就认为完成了写入操作。

就目前而言应用最多和应用最广泛的为协议C。

|  |
| --- |
| **备忘**  两个因素影响你对复制模式的选择：保护和延迟。相比之下吞吐量很大决定了对复制模式的选择。 |

参见“配置你的资源”一节，查看资源配置的复制模式的配置。

**2.4多复制传输**

这个特性在DRBD8.2.7及以后的版本中可用。

DRBD的复制和同步框架套接字层支持多个级别的传输：

**IPv4中的TCP协议：**这是规范的实施方式，也是DRBD默认使用的，可用于任何开启IPv4协议的系统上。

**IPv6的TCP协议：**当配置使用标准的TCP套接字用于复制和同步时，DRBD还可使用IPv6的网络协议。虽然是不同于IPv4，但是这是和使用IPv4的相当的。

**超级套接字：**超级套接字取代TCP/IP的单一堆栈，单片，高效和RDMA能够用套接字实现。DRBD可使用该延迟较低的套接字类型。超级套接字目前必须运行由单一供应商（Dolphin互联解决方案）提供的特定硬件设备上。

**2.5高效同步**

（重新）同步不同于从设备上复制。复制发生于任何对主节点资源进行写时，同步则往往伴随写。相反，它则影响到整个设备。

不管什么理由：主节点失败、次节点失效还是复制链接被中断，只要复制链接被中断同步就是必须的。在这个意义上同步是有效的，DRBD不同步修改块而保持块的顺序写，而维持它的线性秩序，这将会导致如下后果：

**同步速度很快，因为这几个连续的写操作发生一次块的同步。**

**为保持块同步在自然磁盘上的区块分布，同步也是伴随着磁盘磁道的寻找。**

**数据的不一致性，因为在同步的过程中，数据在备用节点上部分过时或者部分被更新，这种数据状态更称为数据不一致性。**

一个节点不能投入使用，从而保证数据的不一致性保持尽量的短。数据同步服务是在主节点上是不间断的以后台进程运行着的。

用下面的公式，你可以简单的估算出同步的时间：

**公式2.1。同步时间**

*tsync*就是预期的同步时间。D是要同步的数据量，你不可能对它有任何的影响的（这个数据由你的应用程序修改而将复制链接打破而产生的），R是同步的速度，这个是可以配置的，通过对复制的限制而控制网络的吞吐量和I/O子系统的。

DRBD的同步效率可进一步提升对数据的消化，也就是校验。当使用校验同步时，不是强制执行覆盖标记不同的块，而是同步他们之前所读取的块，在磁盘上计算出散列的内容。然后比较这个来自对等节点的hash表，如果hash表匹配，则忽略重写该区块。这样就可以大大减少用于解决在断开模式下文件系统重写相同内容的同步时间。

参阅“配置同步速率”和“配置基础校验同步”二节的关于同步配置方面的建议。

**2.6在线设备验证**

这个特性在DRBD8.2.5及以后版本中可用。

在线设备验证，用于可以在节点之间做一个非常有效的方式：逐块逐块得数据完整性检查。

|  |
| --- |
| **备忘**  请注意，“高效”这里是指网络带宽的高效，并保证在任何情况下验证不破坏冗余。在线验证仍是一个资源密集型的操作，会明显提升CPU利用率和负载。 |

它通过一个节点（验证源）对每一个底层的设备的某一资源的块存储设备依次计算出一个加密摘要。接着传输DRBD摘要到对等节点（验证目标），对摘要对应的本地副本块进行验证。如果两个摘要不匹配，标示出块不同进行同步。由于DRBD传输的只是摘要，而不是完整的块，因此对网络资源的要求比较低，从而提高了网络带宽的利用率。

这个过程被称为在线验证，因为它不要求验证的DRBD资源在验证时置为不可用。因此，尽快在线验证需要消耗一部分资源，但是它不会造成服务的中断或者是系统的中断，而且在验证的过程中也不影响后续同步的进行。

在线验证可由常见本地cron调度服务调度在线验证管理运行，比如一周一次或者是一个月一次。

参阅“利用在线设备验证”信息一节，查看如何启用、调用和自动在线验证。

**2.7复制传输完整性验证**

该特性在DRBD8.2.0及以后版本可用。

DRBD使用MD5、SHA-1或者是CRC-32C等加密算法对摘要信息进行终端到终端的信息完整性验证。

|  |
| --- |
| **备忘**  DRBD不提供这些信息摘要的算法，而是由linux的内核的API提供，DRBD只是调用。因此DRBD利用的任何摘要算法需要系统内核提供支持。 |

利用该特性，DRBD对每一个复制到对等节点的数据块都生成信息摘要，对等节点也采用同样的方式对复制的数据块进行完整性验证。如果验证信息不对，则对等节点请求重新发送。这样DRBD复制保护可以放置一些错误的资源，所有这些，如果不检查，将可能导致数据在复制的过程产生脏数据：

**排位错误（“位翻转”）发生在主存储器和网络接口传输数据到对端时（比较常见的一个事实是数据在没有通过TCP校验就被丢到网卡上）**

**位翻转发生在数据从网络接口传送到接受节点的主存储器中时（原因同样是TCP校验丢弃）**

**因为竞争、网络接口固件或者程序错误所产生的任何行形式的脏数据。**

**节点之间的被网络组建组装的位翻转或随机脏数据的注入（如果不使用直连而是后端到后端的连接）**

参阅“配置完整性复制传输检查”一节了解如果启用复制传输完整性检查。

**2.8裂脑通知和自动修复**

在DRBD8.0及更高版本，可实现裂脑自动修复。裂脑自动修复在DRBD0.7中可用，但是主要采取对新的主节点的修改采取抛弃的方式，而且还是不可配置的。默认情况下自动裂脑修复在DRBD中是禁用的。

裂脑通知从DRBD8.2.1之后开始可用。

裂脑大部分情况下是由集群节点间的网络连接临时故障、集群软件管理干预或者是人为错误，导致两个节点都切换为主节点而断开连接。这是一个潜在的有害状态，因为它意味着不能复制数据到对等节点，这样就可能导致两个节点的数据产生分歧，产生不可合并的分裂。

|  |
| --- |
| **备忘**  DRBD的裂脑不同于集群软件的裂脑，如Heartbeat，是分布式集群的通过集群管理软件管理的主机之间失去了所有的连接。本指南采用如下方式加以区分：  **在该段的裂脑指的是DRBD的裂脑。**  **采用集群失去所有连接作为集群裂脑的替代专业术语。** |

当DRBD探测到裂脑时可以设置自动通知（采用邮件或者是其他的方式）。参见“裂脑通知”一节了解更详细的的信息，查看如何配置该特性。

虽然一般情况下建议手工解决裂脑的问题，但是为了彻底的解决裂脑问题，在某些情况下裂脑自动修复还是比较可取的。DRBD有可利用一下方法解决该问题：

**丢弃比较新的主节点的所做的修改。**在这种模式下，当网络重新建立连接并且发现了裂脑，DRBD就会丢弃自切换到主节点后所修改的数据。

**丢弃老的主节点所做的修改。**在这种模式下，DRBD将丢弃首先切换到主节点后所修改的数据。

**丢弃修改比较少的主节点的修改。**在这种模式下，DRBD会检查两个节点的数据，然后丢弃修改比较少的主机上的节点。

**一个节点数据没有发生变化的完美的修复裂脑。**在这种模式下，如果其中一台主机的在发生裂脑时数据没有发生修改，则可简单的完美的修复并声明已经解决裂脑问题。需要注意的是，这几乎是不可能发生的一个情况，即使是两台主机上的只挂载文件系统在DRBD块设备上（设置是只读的），设备的内容也会被修改，也就排除了自动修复的可能性。

|  |
| --- |
| **注意**  **自动裂脑自动修复能不能被接受取决于个人应用。考虑建立一个DRBD的例子库。在“丢弃修改比较少的主节点的修改”兴许对web应用好过数据库应用。与此相反，财务的数据库则是对于任何修改的丢失都是不能容忍的，这就需要不管在什么情况下都需要手工修复裂脑问题。因此需要在启用裂脑自动修复前考虑你的应用情况。** |

参阅“自动修复裂脑策略”一章关于配置DRBD裂脑自动修复的策略的详细信息。

**2.9对磁盘缓冲的支持**

当本地块设备如磁盘或者是RAID逻辑磁盘启用高速写缓存时，一旦将写缓存发回出来就告知磁盘写“完成”。控制管制造商称之为回写模式，而相反的则成为wirte-through。如果在处在回写模式时遇到停电的情况，则可能在发生此之前的回写数据没有来的及写会磁盘，则可能造成数据的丢失。

DRBD使用磁盘刷新来解决这个问题。磁盘刷新是只有在完成相关数据稳定存储的一个写操作，也就是说刷新是有些的写入磁盘而不是缓存。DRVD使用磁盘刷新用于写操作完成其复制数据和原数据。实际上，DRBD绕开写入缓存是非常必要的，如在活动日志更新或隐写后写的执法依赖。这就意味着及时在断电的情况下也能保证其可靠性。

但是比较重要的是，DRBD的需要设置支持才能使用磁盘刷新。在新近比较合理的内核中支持大多数的SCSI和STAT设备的磁盘刷新。Linux软件RAID（md）中的RAID-1也支持磁盘刷新，所有组件所提供的设备也都支持磁盘刷新。这写舍用也同样适用于映射设备（LVM2、DM-raid、多路径）等。

电池供电控制器的高速写缓存（BBWC）使用电池备份他们的挥发性存储。在这些设备上，当电源供电恢复时，在刷新前控制器将电池支持的最近高速写缓存写入磁盘，确保所有的写入由不稳定的存储到稳定的存储上。当这些设备运行在DRBD上时，它可能支持禁用磁盘刷新，从而提高DRBD写入性能。请参阅“禁用备用设备刷新”一些，查阅更详细的信息。

**2.10磁盘错误处理策略**

如果某个节点作为DRBD的后端磁盘设备出现故障，DRBD可能把这个I/O错误传递给上层（通常是文件系统），或者DRBD可能对上层屏蔽了I/O错误。

**Passing on I/O errors：**如果DRBD被配置为 pass on I/O错误，则任何底层设备的错误都会透明地传递给上层I/O层。这样，就由上层来处理错误（这会导致文件系统被重新挂载为read-only）。这个策略不保证服务持续性，并且对大多数用户来说也不推荐。

**Masking I/O errors：**如果DRBD被配置为 detach 底层I/O错误，则DRBD将分离错误。这个I/O错误被DRBD对上层屏蔽，并且DRBD透明地通过网络从对端节点提取受影响的数据块。在这种情况下，DRBD被称为运行在diskless模式，并处理所有相应的I/O操作，读写实际上都是发生在对端（不是本地）。这种diskless运行模式会影响性能，但是服务将继续运行不受影响，并且可以从容地在一个合适的时间迁移到对端节点。（这个方式有点类似 Soft RAID1，当镜像磁盘发生故障时可以确保应用继续运行并提供恢复机会。）

参考配置I/O错误处理策略有关配置I/O处理策略的信息。

**2.11过期数据处理策略**

DRBD区别对待不一致（inconsistent）和过期（outdated）数据。

不一致数据是指不能被正确存取并且使用的数据，一般例子是在某个节点上正在进行同步的数据，此时在这个节点上的数据是部分陈旧的，部分又是更新的。这种情况下，如果这个设备处理一个文件系统，这个文件系统是不能挂载的，甚至不能通过自动的文件系统检查。

过 期数据，相反，是指在第二个节点上的数据是一致的，但是长时间没有和主节点进行同步。这种情况发生在复制链路中断，不论是临时还是永久。数据是过期的，而 且断开的第2个节点被认为是干净的，只是状态反映的是过去的某个时间。为了避免服务使用过期数据，DRBD不允许创建一个过期数据状态的资源。

DRBD有一个接口在一个网络中断时允许一个扩展的应用程序来处理第二节点的过期数据。DRBD然后会拒绝该节点切换为主节点，这样可以保护应用程序不使用过期数据。一个完整实现这个功能的方案是Heartbeat cluster management framework（Heartbeat使用了从DRBD复制链路分离的一个通讯通道）。然而，这个接口是通用的，并且可能适合任何集群管理应用程序。

当一个过期资源重新连接了复制链路，则过期状态将自动清除。一个后台同步过程将自动开始。

the DRBD outdate-peer daemon (dopd)介绍了DRBD/Heartbeat配置保护的案例。

**2.12三路复制**

从DRBD版本8.3.0开始提供三路复制。

当使用三路复制，DRBD对一个已存在的2节点集群增加了一个第三节点，并复制数据到这个第三节点上，使用这个第三节点用于备份和灾难恢复。

三路复制通过在一个已经存在的处理生产数据的堆栈上增加另一个，基于堆栈的DRBD资源，类似以下图示：

这个堆积起来资源是通过异步复制（DRBD protocol A）结合同步复制（DRBD protocol C）来实现的。

三路复制可以永久使用，此时第三节点从生产集群持续更新数据。并且，也可以在需要是建立连接，此时生产集群通常和备份站点断开，周期性地启动site-to-site同步（例如在每天晚上cron任务）。

这个构架适合远程灾备，对于远程镜像可以采用异步方式。由于生产系统不需要等待远程节点完成就可以继续读写，远程站点可以在后续逐步同步好存储系统。

**2.13使用DRBD Proxy实现远距离复制**

DRBD版本8.2.7以上支持DRBD Proxy。

DRBD protocol A是异步实现的，不过写应用程序仍然会在socket输出缓存耗尽后阻塞住（查看 drbd.conf的sndbuf-size选项）。此时，写应用程序将不得不等待直到一些等待数据被缓慢地写入远程系统。

写带宽受限于网络带宽，写入峰值只有在填入有限的socket输出缓存才能有效处理。

可以通过DRBD proxy的缓存机制来解决这个问题。DRBD Proxy会吸收所有从主节点发出的DRBD数据到它的缓存中。DRBD Proxy的缓存大小是任意配置的，只受限于地址空间大小和可用的物理内存。

DRBD Proxy还可以配置在转发数据时是否采用数据压缩和解压缩。DRBD的数据包压缩和解压缩可能可以轻微减少延迟。

建议在多内核SMP系统上使用数据压缩和解压缩。

实际上，多数数据块I/O数据压缩是非常好的，可以有效减轻带宽压力，甚至可以使用DRBD protocol B 和 protocol C。

|  |
| --- |
| **备忘**  **DRBD proxy属于DRBD产品线，并且不是以开源licence发布，需要购买。** |

**2.14基于复制的传输**

Trunk based replication，称为disk shipping，是预处理远程站点的数据服务，通过物理递送方式把硬盘（存储）运送到远程站点。通常适合：

需要复制的数据非常巨大（超过数百G数据）

数据的变化相对较小

站点间的网络带宽有限

在这种情况下如果不采用物理传递硬盘数据的方法，则DRBD初始化同步时间过长，而通过物理传递数据盘，可以大大减少初始化同步时间。

参考Using trunk based replication有关使用案例。

**2.15浮动的对等节点**

从DRBD 8.3.2开始提供浮动对端功能。

一些情况下DRBD需要使用浮动对端配置。在浮动对端配置是，DRBD的对端不是特定命名的主机，而是可以浮动的一系列主机。在这种情况下，DRBD通过IP地址标识对端，而不是主机名。

参考在两个基于SAN后端的Pacemaker集群间配置DRBD复制

**第二篇       安装和配置**

目录

[3. DRBD的编译，安装和配置](http://write.blog.csdn.net/postedit/6460814#_3._DRBD的编译，安装和配置)

[3.1由LINBIT提供的软件包](http://write.blog.csdn.net/postedit/6460814#_3.1由LINBIT提供的软件包)

[3.2由发行商提供的软件包](http://write.blog.csdn.net/postedit/6460814#_3.2由发行商提供的软件包)

[3.2.1SUSE Linux 企业服务器（SLES）](http://write.blog.csdn.net/postedit/6460814#_3.2.1SUSE_Linux_企业服务器（SLES）)

[3.2.2Debian GUN/Linux](http://write.blog.csdn.net/postedit/6460814#_3.2.2Debian_GUN/Linux)

[3.2.3在CentOS](http://write.blog.csdn.net/postedit/6460814#_3.2.3在CentOS)

[3.2.4在Ubuntu](http://write.blog.csdn.net/postedit/6460814#_3.2.4在Ubuntu)

[4 源码编译和安装DRBD](http://write.blog.csdn.net/postedit/6460814#_4_源码编译和安装DRBD)

[4.1下载DRBD源码](http://write.blog.csdn.net/postedit/6460814#_4.1下载DRBD源码)

[4.2从DRBD源码库中签出源码](http://write.blog.csdn.net/postedit/6460814#_4.2从DRBD源码库中签出源码)

[4.3从源码编译DRBD](http://write.blog.csdn.net/postedit/6460814#_4.3从源码编译DRBD)

[4.3.1检查编译的条件](http://write.blog.csdn.net/postedit/6460814#_4.3.1检查编译的条件)

[4.3.2准备内核源码树](http://write.blog.csdn.net/postedit/6460814#_4.3.2准备内核源码树)

[4.3.3准备DRBD的编译树](http://write.blog.csdn.net/postedit/6460814#_4.3.3准备DRBD的编译树)

[4.4DRBD的rpm包的创建](http://write.blog.csdn.net/postedit/6460814#_4.4DRBD的rpm包的创建)

[4.5创建DRBD的debian包](http://write.blog.csdn.net/postedit/6460814#_4.5创建DRBD的debian包)

[5 配置drbd](http://write.blog.csdn.net/postedit/6460814#_5_配置drbd)

[5.1准备底层存储](http://write.blog.csdn.net/postedit/6460814#_5.1准备底层存储)

[5.2准备网络配置](http://write.blog.csdn.net/postedit/6460814#_5.2准备网络配置)

[5.3资源的配置](http://write.blog.csdn.net/postedit/6460814#_5.3资源的配置)

[5.3.1配置样例](http://write.blog.csdn.net/postedit/6460814#_5.3.1配置样例)

[5.3.2global 部分](http://write.blog.csdn.net/postedit/6460814#_5.3.2global_部分)

[5.3.3Common部分](http://write.blog.csdn.net/postedit/6460814#_5.3.3Common部分)

[5.3.4Resource部分](http://write.blog.csdn.net/postedit/6460814#_5.3.4Resource部分)

[5.4首次启用资源](http://write.blog.csdn.net/postedit/6460814#_5.4首次启用资源)

[5.5初始化设备同步](http://write.blog.csdn.net/postedit/6460814#_5.5初始化设备同步)

[5.6基于复制的传输](http://write.blog.csdn.net/postedit/6460814#_5.6基于复制的传输)

**3. DRBD的编译，安装和配置**

**3.1由LINBIT提供的软件包**

LINBIT是赞助DRBD项目的公司，为商业用户提供DRBD的二进制包。这些包在<http://www.linbit.com/support/>中可以找到，被认为是“官方”DRBD的基础。

它的发行版本如下：

这些支持以下发行版本：

**红帽企业的linux（RHEL），版本4和5**

**SUSE Linux企业操作系统（SLES），版本9、10和11**

**Debian GUN/linux操作系统，版本4.0（etch）和5.0（lenny）**

**Ubuntu Server Edition LTS，版本6.06（Dapper Drake）和8.04（Hardy Heron）**

LINBIT发布所有DRBD的新源码包。

基于RPM包（SLES、RHEL）的安装可以通过简单的调用rpm -i （新安装）或者是rpm –U （升级）对DRBD进行安装。

对于基于Debian的系统（GUN/Linux，Ubuntu），Drbd8-utils和Drbd8-module 包的安装使用dpkg -i 或者是gdebi（如果可用的话）。

**3.2由发行商提供的软件包**

相当数量的发布包包括DRBD，包括二进制的预编译包。支持这些编译，如果有的话就是由发行商提供相关发布，他们释放的周期往往落后于DRBD源码的发行。

**3.2.1SUSE Linux 企业服务器（SLES）**

**在SLES7和10中包括DRBD0.7，在SLES11高可用扩展中包含DRBD8.2。**

**在SLES，DRBD通常是通过安装软件的安装组件YaST2,捆绑在“高可用”软件包中供选择。**

**用户通常使用简单的命令行安装：**

yast -i drbd或者rug install drbd

**3.2.2Debian GUN/Linux**

**自动5.0（Lenny）中包含DRBD8，从Debian 3.1（sarge）中包含DRBD0.7。**

**在Lenny中（现在已将DRBD预编译在内核中，不在需要使用module-assistan），可以通过如下安装DRBD：**

apt-get install drbd8-utils drbd8-module

**在Debian3.1和4.0，则必须通过如下命令安装：**

apt-get install drbd0.7-utils drbd0.7-module-source build-essential module-assistant

module-assistant auto-install drbd0.7

请参阅“创建DRBD的Debian包”一节，有关涉及安装以及module-assistant的详细资料。

**3.2.3在CentOS**

**CentOS 5中包含DRBD8，CentOS 4中包含DRBD0.7**

**DRBD可以通过yum命令进行安装（需要注意的是提要支持软件仓库）**

yum install drbd kmod-drbd

**3.2.4在Ubuntu**

**Ubuntu7.10（gusty gibbon）包含DRBD8，Ubuntu6.06（Dapper Drake）中包含DRBD0.7。可以通过激活Ubuntu中的镜像/etc/apt/sources.list，执行如下命令获取DRBD：**

apt-get update

apt-get install drbd8-utils drbd8-module-source build-essential module-assistant

module-assistant auto-install drbd8

|  |
| --- |
| **警告**  **Ubuntu 6.10（Edgy EFT）和7.04（Feisty Fawn）都包含DRBD8的预发行版，这是从未在生产系统中正式使用的版本。Ubuntu也包含DRBD0.7，但是生产可适当的用（但是版本已经过期）。** |

**4 源码编译和安装DRBD**

**4.1下载DRBD源码**

DRBD的历史版本和当前版本的压缩包都可以从<http://oss.linbit.com/drbd/>上下载。按照惯例，DRBD的源码包以drbd-x.y.z的方式命名，其中想，x,y,z对应的为主、辅和错误的修正版本号。

DRBD压缩包的大小不超过0.5M。要下载并解压到当前工作目录，可使用如下命令：

wget http://oss.linbit.com/drbd/8.3/drbd-8.3.4.tar.gz

tar -xzf drbd-8.3.4.tar.gz

|  |
| --- |
| **备忘**  **上面wget下载的源码只是一个例子，当然你可以下载你喜欢的版本。** |

建议将DRBD放到存放源代码的普通目录中，如/usr/src或者是/usr/local/src，本指南中假定在/usr/src目录中。

**4.2从DRBD源码库中签出源码**

DRBD的源码保存在一个公共的Git库中，可以在[http://git.drbd.org/上浏览。要从库中签出DRBD](http://git.drbd.org/%E4%B8%8A%E6%B5%8F%E8%A7%88%E3%80%82%E8%A6%81%E4%BB%8E%E5%BA%93%E4%B8%AD%E7%AD%BE%E5%87%BADRBD)的源码，就必须将DRBD释放到库中。这本例中签出的为DRBD8.3：

git clone git://git.drbd.org/drbd-8.3.git

如果您的防火墙不允许TCP连接到9418端口，您也可以通过HTTP签出（请注意，使用http协议要比Git签出的方式慢的多，因此建议尽量的使用Git签出）。

git clone http://git.drbd.org/drbd-8.3.git

先创建一个命名为drbd-8.3的Git的签出目录，也就是将命令的文件夹的名字和签出的源代码的名字一致，可使用以下命令：

cd drbd-8.3

git checkout drbd-8.3.x

这里的X就代表你想签出的DRBD的建立点。

签出的目录是和解压后的指定版本的源代码一样的，也就就是使用DRBd的源代码了。

|  |
| --- |
| **备忘**  **实际上源代码解压和git签出两个来源之间还是有些细微的差别的：**  **git签出包含一个debian/subdirectoy，而源代码解压却没有。这是因为debian版本的维护者要求加入的。**  **源代码的tarball包含帮助手册，而git不包含。虽然docbook不是源码必须的，但是git签出需要一个完整的docbook手册的地址。** |

**4.3从源码编译DRBD**

**4.3.1检查编译的条件**

在编译DRBD的源码前，机器必须符合一下条件：

**make，gcc：glibc的开发包，是flex scanner generator必须安装的。**

|  |
| --- |
| **备忘**  **必须保证使用的gcc编译模块运行在内核上，如果系统中有多个版本的gcc，需要选择特定版本的gcc。** |

**如果直接从git签出的文件中编译，guncutoconf也是必须的。这个如果从源码编译这是不需要的。**

**如果选择运行一个stock kernel，就需要安装一个和之相匹配的预编译的内核头文件包。这些通常命名为kernel-dev、kernel-headers、linux-headers或者类似的名字。这种情况下，可以跳过“准备内核源码树”一节，而进行“准备DRBD的编译树”一节。**

**如果选择的不是stock kernel的（比如：系统运行在一个从源码编译配置的内核上），那么内核的源文件则必须安装，则可编译安装和内核源代码版本匹配或者是相似的软件包。**

|  |
| --- |
| **备忘**  **在以rpm为基础的系统上，这些软件包命名类似于kernel-source-version.rpm的形式，这就很容易和kernel-version.src.rpm相混淆。而前者则是正确的DRBD编译安装包。** |

在kernel.org上“Vanlilla”内核包命名的格式为linux-version-tar.bz2并应该被解压到/usr/src/linux-version中，使用连接的方式指向/usr/src/linux的文件夹。

在这种情况下，编译DRBD对应的内核源码，则必要进行“准备内核源码树”一节。

**4.3.2准备内核源码树**

必须在解压后的内核源码所在的目录准备建立DRBD的源代码树，通常情况下，这个目录为/usr/src/linux-version，或者是就是一个连接名为/usr/src/linux的目录。

cd /usr/src/linux

接下来的为建议的步骤而不是必须的。务必复制现有的.config文件到一个安全的位置，在这一步的基础上初始化内核源码树，消除之前编译和配置的影响：

make mrproper

接下来就可以复制当前运行的配置文件到内核源码树，下面有些选项需要选择：

**通过/proc将最近内核的合理配置输出到当前正在运行的内核配置文件中，可以进行复制，如下：**

zcat /proc/config.gz > .config

**在SUSE的内核上包含一个cloneconfig target，所在这这类系统上，操作如下：**

make cloneconfig

**一些安装将内核配置文件安装到/boot下，这就需要：**

cp /boot/config-`uname -r` .config

**最后，你可以将已经配置好的文件拷贝到当前运行的内核中运行。**

**4.3.3准备DRBD的编译树**

任何DRBD的编译都需要先配置DRBD的源码树和脚本：

|  |
| --- |
| **备忘**  **本节中所提到的适用于DRBD8.3.6及以上的版本。一直到DRBD8.3.5还没有配置脚本。**  **当编译git签出的源码时，配置脚本也是不存在的，因此必须先签出它才行。** |

可以使用—help查看配置帮助文件列表。在下面的表格中总结了些比较重要的选项：

表4.1 DRBD所支持的配置选项

| **选项** | **描述** | **默认** | **备注** |
| --- | --- | --- | --- |
| --prefix | 安装路径的选项 | /usr/local | 这是默认的文件系统标准的无包装软件的安装路径，包装软件一般覆盖/usr |
| --localstatedir | 目录状态定位路径 | /usr/local/var | 即使有默认的prefix，但大多数用户还是想重写/var |
| --sysconfdir | 系统配置路径 | /usr/local/etc | 即使有默认的prefix，但大多数用户还是想在/etc下 |
| --with-km | 编译DRBD的内核模块 | 无 | 在编译DRBD内核模块时开启此选项 |
| --with-utils | 编译DRBD集成userland工具 | yes | 当编译一个新的内核版本的DRBD，而不同时升级DRBD，禁用次选项 |
| --with-heartbeat | 编译DRBD集成 Heartbeat | yes | 如果不适用DRBD的heartbeat v1 资源代理或dopd，则可禁用该选项 |
| --with-pacemaker | 编译DRBD集成 Pacemaker | yes | 如不打算启用Pacemaker功能，则可禁用该选项 |
| --with-rgmanager | 编译DRBD集成 红帽集群套件 | no | 如想适用DRBD整合红帽几圈套件的rgmanager，这需要启用该选项 |
| --with-xen | 编译DRBD集成Xen | yes (在X86平台上) | 如果不打算集成Xen模块，则可禁用该选项 |
| --with-bashcompletion | 编译可编程的bash**drbdadm** | yes | 如果使用其他的bash的shell，或者不想利用可编程的drbdadm，这可禁用该选项 |
| --enable-spec | 创建一个rpm spec文件 | no | 仅对包的编译者：如果想创建一个rpm spec，可启用该选项。见“编译DRBD的RPM”一节 |

配置脚本将建立使用符合需求的DRBD。它通过自动检测所调用的参数对DRBD进行配置，因此在配置相应的参数时，要小心重写默认的选项。

配置脚本在被调用的目录中创建一个日志文件名为config.log。在记录编译时的问题时，也比较智能话的记录整个编译的过程。

**4.4DRBD的rpm包的创建**

|  |
| --- |
| **备忘**  **本节中所提到的适用于DRBD8.3.6及以上的版本。一直到DRBD8.3.5还是用不同的rpm安装的方法。** |

DRBD的安装包含一个直接以安装rpm出DRBD的源代码树体系。为创建rpm包，这部分叫做“检查创建的前提条件”适用于以上同样make安装的方式，除非你还需要rpm的创建工具。

另外，如果你是创建与内核中可用的预编译头的方式，可以参见“准备内核源代码树”一节。

有两种方案创建rpm包。简单的方法是简单在顶层makefile调用rpm目标文件：

$ ./configure

$ make rpm

$ make km-rpm

这种方法是将自动从预先定义的模板生成spec文件，然后利用这些模板文件，建立二进制的rpm包。

生成Rpm的方式生成大量的rpm包：

**表4.2。 DRBD技术的userland RPM包**

| **包名** | **描述** | **依赖关系** | **备注** |
| --- | --- | --- | --- |
| drbd | DRBD 元软件包 | 其他所有的 drbd-\* 软件包 | 顶层的虚拟包。安装时，在所有其他的userland作为依赖包中提取。 |
| drbd-utils | 二进制管理使用程序 | DRBD启用需要 |  |
| drbd-udev | udev的集成设施 | drbd-utils, udev | 启用udev来管理用户友好连结到DRBD设备 |
| drbd-xen | Xen DRBD helper scripts | drbd-utils, xen | Enables xend to auto-manage DRBD resources |
| drbd-heartbeat | DRBD Heartbeat integration scripts | drbd-utils, heartbeat | Enables DRBD management by legacy v1-style Heartbeat clusters |
| drbd-pacemaker | DRBD Pacemaker integration scripts | drbd-utils, pacemaker | Enables DRBD management by Pacemaker clusters |
| drbd-rgmanager | DRBD Red Hat Cluster Suite integration scripts | drbd-utils, rgmanager | Enables DRBD management by rgmanager, the Red Hat Cluster Suite resource manager |
| drbd-bashcompletion | Progammable bash completion | drbd-utils, bash-completion | Enables Programmable bash completion for the drbdadm utility |

另外一种是更灵活的方法是配置并生成spec文件，你可以进行任何你所需要的更改，然后使用rpmbuild命令生成：

$ ./configure --enable-spec

$ make tgz

$ cp drbd\*.tar.gz `rpm -E \_sourcedir`

$ rpmbuild -bb drbd.spec

如果你要建立drbd的公用用户空间和内核模块的rpm，使用：

$ ./configure --enable-spec --with-km

$ make tgz

$ cp drbd\*.tar.gz `rpm -E \_sourcedir`

$ rpmbuild -bb drbd.spec

$ rpmbuild -bb drbd-km.spec

这种方式将被创建在rpm 配置（或个人的rpmmacros配置）的地方

当创建这些软件包后，你可以安装、升级和卸载系统中的任何其他的rpm包。

请注意任何内核的升级都需要生成心的drbd-km包以匹配新的内核。

当升级drbd到一个新版本时，只需要创建新的drbd userland 包。然而当升级新的内核和新的drbd版本时，则需要同时升级包。

**4.5创建DRBD的debian包**

DRBD创建系统包含一个直接从DRBD源代码树建立出的Debian包。可参见“检查建立的前提条件”一节建立debian软件包，除了需要dpkg-dev包含debian软件包工具和fakeroot包（如果想非root用户建立DRBD—强烈推荐），基本上和建立并使用make安装的方式是一样的。

另外，如果你是创建与内核中可用的预编译头的方式，可参见“准备内核源代码树”一节。

DRBD源代码书包含一个debian软件包所需文件的debian子目录。但是这个子目录并不在DRBD的tar包内，因此如果需要时则需要创建一个特定的git，将其签出。

可以使用一下命令签出DRBD的debian软件包：

dpkg-buildpackage -rfakeroot -b -uc

|  |
| --- |
| **备忘**  **如编译drbd的软件包使用非root用户（-rfakeroot）以二进制的方式（-b），并禁用变更文件的签名。其他的编译选项可以参见dpkg-buildpackage手册。** |

这个创建的过程将创建两个debian的软件包：

1.     命名为drbd8-utils\_x.y.z-BUILD\_ARCH.deb，drbd的用户控件工具

2.     命名为drbd8-module-source\_x.y.z-BUILD\_all。Deb，源代码匹配包。

当你创建这些软件包后，就可以安装、升级或者是卸载他们，就像其他的Debian软件包一样。

通过debian的模块儿辅助功能，从源码包中编译和安装drbd的实际的内核模块儿是比较容易的：

module-assistant auto-install drbd8

也可以使用上面命令的简写模式：

m-a a-i drbd8

需要注意的是任何内核的升级都需要重新编译其内核模块儿（如模块辅助）来匹配新的内核。相比之下Drbd8-utils和drbd8-module-source 源码包，则只需要升级到新的drbd版本。无论是升级到新的内核还是新的drbd版本，都需要升级以上两个软件包。

**5 配置drbd**

**5.1准备底层存储**

安装过drbd之后，就必须在集群的两个节点上预留一个大小一致的存储区域。这将是drbd的底层的设备。可以使用系统中任何型号的块设备达到次目的。典型的例子包括：

**一个磁盘分区（或一个完成的物理磁盘）**

**软RAID设备**

**LVM逻辑卷或者其他有linux的device-mapper基础块设备**

**EVMS**

**服务器上的其他任何类型的块设备。DRBD8.3及以上的版本，也可以使用资源堆放，也就意味着可用一个drbd设备为其他的一个底层的设备，一些特殊的也可以考虑资源堆放，可参见“创建一个三节点设置”了解详细的配置介绍。**

|  |
| --- |
| **备忘**  **DRBD虽然也可以使用loop设备作为底层的设置，但是因为其有死锁的问题，因此并不推荐。** |

创建drbd资源前并不要求该存储区是空的。事实上，它是一种常见从以前的两个非冗余单节点的服务器，使用drbd创建两个节点的集群。（如打算这样做的话，请参阅“DRBD源数据”一节，了解一些注意事项。）

在本指南中，我们假设为一个很简单的设置：

**两台主机都有空闲的分区/dev/sda7**

**我们正在使用内部元数据**

**5.2准备网络配置**

建议运行DRBD复制在专用链接环境下，虽然没有严格的要求。本指南建议最合理的选择为直接的、千兆G太网络连接。如果你通过交换里连接DRBD，建议使用冗余组建和linux绑定却东程序（主动备份模式）。

一般情况下不建议通过路由器运行drbd复制，因为通过路由器的方式表现出明显的缺点（影响吞吐量并造成延迟）。

考虑本地防火墙的因素，drbd建议采用单独的、可配置TCP7788以上的端口，但不变的是TCP端口上侦听的TCP资源。DRBD使用两个单独的TCP连接（任何方向之一）对资源进行配置。对于正确的使用DRBD功能，必须让防火墙运行这些连接的进行。

比防火墙更安全的方式为强制访问控制（MAC），如启动selinux或者是appmrmor。因此需要调整本地的安全策略来适应DRBD功能的运行。

自然需要保证DRBD所使用的端口没有被其他的程序所占用。

|  |
| --- |
| **备忘**  **DRBD资源不可能支持多个TCP连接。如果想让DRBD连接实现负载均衡或者是冗余，建议在比较容易实现的以太网级别进行（再次使用bongding设备）。** |

在本指南中，我们假设为一个很简单的设置：

**两个DRBD主机分别有一个未启用的网络接口eth1，给他他们分别配置ip地址10.1.1.31和10.1.1.32.**

**在两台主机上都没有其他程序占用TCP端口7788和7799**

**本地防火墙配置允许两台机器通过这些TCP端口的进和出**

**5.3资源的配置**

DRBD的所有的控制都是在配置文件/etc/drbd.conf中。通常情况下配置文件包含如下内容：

include "/etc/drbd.d/global\_common.conf";

include "/etc/drbd.d/\*.res";

通常情况下，/etc/drbd.d/global\_common.conf包含global和common的DRBD配置部分，而.res文件都包含一个资源的部分。

在一个单独的drbd.conf文件中配置全部是可以实现的，但是占用的配置很快就会变得混乱，变得难以管理，这也是为什么多文件管理作为首选的原因之一。

无论采用哪种方式，需必须保持在各个集群节点的drbd.conf以及其他的文件完全相同。

在drbd的源代码包的脚本子目录中包含配置文件的样例。如二进制安装包直接将其安装在/etc或者是包中特定的样例目录/usr/share/doc/packages/drbd中。

|  |
| --- |
| **备忘**  **DRBD资源不可能支持多个TCP连接。如果想让DRBD连接实现负载均衡或者是冗余，建议在比较容易实现的以太网级别进行（再次使用bongding设备）。** |

本节只介绍这些为了保证drbd运行的必须了解的配置文件相关的方面。配置文件的语法和内容都相继的记录在drbd.conf（5）中。

**5.3.1配置样例**

在本指南中，我们假设为一个很简单的设置：

global {

  usage-count yes;

}

common {

  protocol C;

}

resource r0 {

  on alice {

    device    /dev/drbd1;

    disk      /dev/sda7;

    address   10.1.1.31:7789;

    meta-disk internal;

  }

  on bob {

    device    /dev/drbd1;

    disk      /dev/sda7;

    address   10.1.1.32:7789;

    meta-disk internal;

  }

}

该配置样例做一下详解：

**选择参与drbd的使用情况统计（见下文）**

**资源配饰使用完全同步复制协议（Protocol C），除非另有明确指定。**

**集群包含两个节点。Alice和Bob**

**资源命名为r0，并使用底层设备/dev/sda7，并与内部原数据配置资源**

**资源使用网络TCP连接端口7789，并分别绑定在ip地址10.1.1.31和10.1.1.32上**

**5.3.2global 部分**

在配置文件中这部分只允许出现一次，他通常在/etc/drbd.d/global\_common.conf文件中。在单一的配置文件中，他应该置顶。在本节提供的几个选项中，只有一个是大多数用户相关。

usage-count.DRBD用于统计应用各个版本的信息。当新的版本的drbd被安装就会和http server进行联系。当然也可以禁用该选项，默认情况下是启用该选项的。

|  |
| --- |
| **备忘**  **DRBD使用统计数据是公开的，可以参见http://usage.drbd.org** |

**5.3.3Common部分**

该本分提供一个快捷的方式来定义配置每个可继承的资源，它通常在/etc/drbd.d/gloabal\_common.conf中。可以在定义的每一个资源的基础上定义任意选项。

如果使用多个资源的话，该部分并不是严格要求但是是强烈建议的。否则的话配置文件很快就会变的让人难以理解了。

在上面的例子中，我们使用Protocol C，所以在每一个资源配置（包括r0）基础该选项，除非你明确配置使用另外的协议选项。对其他的同步协议选项，可以参见“复制模式”一节。

**5.3.4Resource部分**

每个资源的配置通常都命名为/etc/drbd.d/resource.res。所有的drbd资源必须通过指定的配置方式进行命名，可以使用US-ASCii内的除空格外的任意的标示符。

每一个资源配置都必须有两个主机上的分节（一个节点上一个）。

所以的配置设置都继承共同的部分（如果存在的话），或者是从DRBD默认设置中继承。

事实上，你也可以使用主机分节上资源部分的一个速记符号，也可以指定每个选项的值在量主机上一样。这样，和上面的例子比我们可以进一步压缩该部分：

resource r0 {

  device    /dev/drbd1;

  disk      /dev/sda7;

  meta-disk internal;

  on alice {

    address   10.1.1.31:7789;

  }

  on bob {

    address   10.1.1.32:7789;

  }

}

这个符号在drbd8.2.1及以上版本可用。

**5.4首次启用资源**

如上一节所述完成对资源的配置后即可启用资源。

|  |
| --- |
| **备忘**  **以下步骤必须在两个节点上完成** |

1.     创建元数据设备。这步只有先创建设置然后才能初始化设备：

drbdadm create-md resource

v08 Magic number not found

Writing meta data...

initialising activity log

NOT initialized bitmap

New drbd meta data block sucessfully created.

success

2.     附加到备份设置。这步将drbd资源和后端设备连接

drbdadm attach resource

3.     设置同步参数。这步设置drbd资源的同步参数

drbdadm syncer resource

4.     4连接对等节点。这一步连接对等节点的资源

drbdadm connect resource

|  |
| --- |
| **提示**  **必须注意操作的步骤为drbdadm attach、drbdadm syncer 、drbdadm connect，不然可能导致drbd崩溃。** |

5.     查看/proc/drbd。/proc/drbd 为Drbd的虚拟状态文件，包含如下类似的信息：

cat /proc/drbd

version: 8.3.0 (api:88/proto:86-89)

GIT-hash: 9ba8b93e24d842f0dd3fb1f9b90e8348ddb95829 build by buildsystem@linbit, 2008-12-18 16:02:26

 1: cs:Connected ro:Secondary/Secondary ds:Inconsistent/Inconsistent C r---

    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:200768

这两个节点上出现了Inconsistent/Inconsistent（不一致/不一致）的状态。

到这里，drbd已经成功的分配好磁盘和网络资源，为运行做好了准备。但是他还不知道使用那个节点作为初始化设备的同步源。

**5.5初始化设备同步**

Drbd的运行还需要以下两个以下的步骤：

1.     选择一个初始同步源。如果是新初始化的或者是空盘时，这个选择是任意的。但是如果其中的一个节点已经在使用并包含有用的数据，那么选择哪个节点同步源就是至关重要的。如果选择了错误的初始化同步方向，就会造成数据丢失，因此需要十分的小心。

2.     启动初始化化完全同步。这一步只能在初始化资源配置的一个节点上进行，并作为同步源选择的节点上。发出如下命令支持此步骤：

        drbdadm -- --overwrite-data-of-peer primary resource

执行此命令后，初始化同步开始进行。可以通过查看/proc/drbd检测同步的进展情况。同步的时间和设备的大小有一定的关系。

到现在为止，drbd的设置已经全面投入使用，甚至是在初始化同步完毕之前（这期间性能可能有所降低）。现在可以创建一个文件系统作为raw的存储块设备，然后挂载它，之后就可执行任何的其他的操作了。

现在可进行第6章，了解常见的任务管理方面的知识，这部分将描述常见的资源的管理任务。

**5.6基于复制的传输**

为跳过初始化设备同步，为远程节点预置数据并保持同步，请参照以下步骤进行操作。

|  |
| --- |
| **备忘**  假设本地节点已经配置，但是主节点的drbd资源断开了连接。也就是说设备配置已经完成，相同的配置文件drbd.conf存在在两个对等的节点上，也发出了进行初始化本地节点的命令，但是远程节点尚未连接。 |

1．在本地节点发出如下命令：

drbdadm new-current-uuid --clear-bitmap <resource>

2．创建一个一致的、逐字拷贝的数据的资源以及其元数据。例如，从raid1中删除一个支持热插拔的磁盘，然后将其更换为一个新的磁盘，并重建raid以保证持续的冗余。被删除未一个可以撤掉的逐字复制的设备。如果本地的磁盘支持快照副本（如基于LVM的DRBD），你可以使用dd创建一个逐位复制的快照。

3．在本地节点的问题：

drbdadm new-current-uuid resource

请注意，这里没有调用 --clear-bitmap选项

4．副本物理传输到远程对等节点。

5．新增副本到远程节点。这可能会再次造成物理磁盘的堵塞，逐位拷贝磁盘现有的数据到远程节点上。

一定要保证恢复或者是拷贝的数据不仅仅是复制的数据，而且还需要相关的drbd的元数据。如果没有这样做的话，磁盘的传输是没有意义的。

6．启用远程节点的资源：

drbdadm up resource

当两个对等节点连接后，不会启动进行完整的同步，相反，自动同步只包含哪些因为drbdadm调用而改变的块-- --clear-bitmap new-current-uuid.

即使自此之后没有发生任何的变化，由于涉及活动日志的更新等，也有可能出现一个简短的同步期。这些可以通过使用基于校验的同步而减轻同步。

|  |
| --- |
| **提示**  **无论资源是普通的资源还是堆放的资源，都可以使用相同的过程。对于堆放的DRBD资源，需要通过启动参数 –S或者是 –stacked选项。** |

**第三篇 利用drbd工作**

**6 常见的管理任务**

目录

[6.1 检查DRBD的状态](http://write.blog.csdn.net/postedit/6460814#_6.1_检查DRBD的状态)

[6.1.1通过drbd-overview查看DRBD的状态](http://write.blog.csdn.net/postedit/6460814#_6.1.1通过drbd-overview查看DRBD的状态)

[6.1.2从/proc/drbd中查看状态信息](http://write.blog.csdn.net/postedit/6460814#_6.1.2从/proc/drbd中查看状态信息)

[6.1.3连接状态](http://write.blog.csdn.net/postedit/6460814#_6.1.3连接状态)

[6.1.4资源角色](http://write.blog.csdn.net/postedit/6460814#_6.1.4资源角色)

[6.1.5 磁盘状态](http://write.blog.csdn.net/postedit/6460814#_6.1.5_磁盘状态)

[6.1.6I / O的状态标志](http://write.blog.csdn.net/postedit/6460814#_6.1.6I_/_O的状态标志)

[6.1.7绩效指标](http://write.blog.csdn.net/postedit/6460814#_6.1.7绩效指标)

[6.2启用和禁用资源](http://write.blog.csdn.net/postedit/6460814#_6.2启用和禁用资源)

[6.2.1启用资源](http://write.blog.csdn.net/postedit/6460814#_6.2.1启用资源)

[6.2.2禁用资源](http://write.blog.csdn.net/postedit/6460814#_6.2.2禁用资源)

[6.3重新配置资源](http://write.blog.csdn.net/postedit/6460814#_6.3重新配置资源)

[6.4升级和降级资源](http://write.blog.csdn.net/postedit/6460814#_6.4升级和降级资源)

[6.5启用双主模式](http://write.blog.csdn.net/postedit/6460814#_6.5启用双主模式)

[6.5.1永久双主模式](http://write.blog.csdn.net/postedit/6460814#_6.5.1永久双主模式)

[6.5.2临时双主模式](http://write.blog.csdn.net/postedit/6460814#_6.5.2临时双主模式)

[6.5.3开机自动启用](http://write.blog.csdn.net/postedit/6460814#_6.5.3开机自动启用)

[6.6在线设备验证](http://write.blog.csdn.net/postedit/6460814#_6.6在线设备验证)

[6.6.1启用在线设备验证](http://write.blog.csdn.net/postedit/6460814#_6.6.1启用在线设备验证)

[6.6.2调用在线核查](http://write.blog.csdn.net/postedit/6460814#_6.6.2调用在线核查)

[6.6.3自动在线验证](http://write.blog.csdn.net/postedit/6460814#_6.6.3自动在线验证)

[6.7配置同步率](http://write.blog.csdn.net/postedit/6460814#_6.7配置同步率)

[6.7.1固定的同步速率配置](http://write.blog.csdn.net/postedit/6460814#_6.7.1固定的同步速率配置)

[6.7.1临时固定的同步速率配置](http://write.blog.csdn.net/postedit/6460814#_6.7.1临时固定的同步速率配置)

[6.8配置基于校验的同步](http://write.blog.csdn.net/postedit/6460814#_6.8配置基于校验的同步)

[6.9配置拥堵策略和暂停复制](http://write.blog.csdn.net/postedit/6460814#_6.9配置拥堵策略和暂停复制)

[6.10配置I/O错误处理策略](http://write.blog.csdn.net/postedit/6460814#_6.10配置I/O错误处理策略)

[6.11配置复制传输完整性验证](http://write.blog.csdn.net/postedit/6460814#_6.11配置复制传输完整性验证)

[6.12重新调整资源](http://write.blog.csdn.net/postedit/6460814#_6.12重新调整资源)

[6.12.1在线增长](http://write.blog.csdn.net/postedit/6460814#_6.12.1在线增长)

[6.12.2离线增长](http://write.blog.csdn.net/postedit/6460814#_6.12.2离线增长)

[6.12.3在线收缩](http://write.blog.csdn.net/postedit/6460814#_6.12.3在线收缩)

[6.12.4离线收缩](http://write.blog.csdn.net/postedit/6460814#_6.12.4离线收缩)

[6.13禁用后端设备刷新](http://write.blog.csdn.net/postedit/6460814#_6.13禁用后端设备刷新)

[6.14配置裂脑行为](http://write.blog.csdn.net/postedit/6460814#_6.14配置裂脑行为)

[6.14.1裂脑通知](http://write.blog.csdn.net/postedit/6460814#_6.14.1裂脑通知)

[6.14.2 裂脑自动修复策略](http://write.blog.csdn.net/postedit/6460814#_6.14.2_裂脑自动修复策略)

[6.15创建三节点设置](http://write.blog.csdn.net/postedit/6460814#_6.15创建三节点设置)

[6.15.1设备堆叠的注意事项](http://write.blog.csdn.net/postedit/6460814#_6.15.1设备堆叠的注意事项)

[6.15.2配置堆叠的资源](http://write.blog.csdn.net/postedit/6460814#_6.15.2配置堆叠的资源)

[6.15.3启用堆叠资源](http://write.blog.csdn.net/postedit/6460814#_6.15.3启用堆叠资源)

[6.16使用DRBD proxy](http://write.blog.csdn.net/postedit/6460814#_6.16使用DRBD_proxy)

[6.16.1DRBD proxy部署注意事项](http://write.blog.csdn.net/postedit/6460814#_6.16.1DRBD_proxy部署注意事项)

[6.16.2安装](http://write.blog.csdn.net/postedit/6460814#_6.16.2安装)

[6.16.3License 文件](http://write.blog.csdn.net/postedit/6460814#_6.16.3License_文件)

[6.16.4配置](http://write.blog.csdn.net/postedit/6460814#_6.16.4配置)

[6.16.5控制DRBD proxy](http://write.blog.csdn.net/postedit/6460814#_6.16.5控制DRBD_proxy)

[6.16.6故障排除](http://write.blog.csdn.net/postedit/6460814#_6.16.6故障排除)

**6.1 检查DRBD的状态**

**6.1.1通过drbd-overview查看DRBD的状态**

查看drbd状态最方便的方式是通过drbd-overview查看。

drbd-overview

  0:home                 Connected Primary/Secondary

  UpToDate/UpToDate C r--- /home        xfs  200G 158G 43G  79%

  1:data                 Connected Primary/Secondary

  UpToDate/UpToDate C r--- /mnt/ha1     ext3 9.9G 618M 8.8G 7%

  2:nfs-root             Connected Primary/Secondary

  UpToDate/UpToDate C r--- /mnt/netboot ext3 79G  57G  19G  76%

**6.1.2从/proc/drbd中查看状态信息**

/proc/drbd是一个虚拟的文件，用于显示当前配置的所有drbd资源的实时状态，可是使用以下命令查看该文件的内容：

cat /proc/drbd

version: 8.3.0 (api:88/proto:86-89)

GIT-hash: 9ba8b93e24d842f0dd3fb1f9b90e8348ddb95829 build by buildsystem@linbit, 2008-12-18 16:02:26

 0: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r---

    ns:0 nr:8 dw:8 dr:0 al:0 bm:2 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0

 1: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r---

    ns:0 nr:12 dw:12 dr:0 al:0 bm:1 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0

 2: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r---

    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0

第一行显示的为当前系统使用的drbd的版本，第二行包含一些特定的编译信息。

在本例中的其他行为重复显示的每一个drbd设备的配置信息，设备的次要编号。在这里是0，对应的设备则为/dev/drbd0。

/proc/drbd包含了其他的一些具体的各种资源的信息：

CS（connnection state 连接状态）。网络连接状态，可参见“6.1.3”章节查看有关各种连接状态的详细信息。

RO（role 角色）。节点的角色。本地节点的角色显示在前，斜线后面的为对等节点的角色。详细可参阅“6.1.4”章节有关资源角色的详细信息。

DS（disk stats 磁盘状态）。磁盘状态，斜线前的为本地节点的磁盘状态，斜线后的为对等节点的磁盘状态。详细可参阅“6.1.5”章节有关各种磁盘状态的详细信息。

Replication protocol 复制协议。资源使用的复制协议，不外乎A/B/C。详细可参阅“2.3”节有关复制协议。

I/O标示。六个状态标示反映出资源的I/O状态。详细可参见“6.1.6”章节了解有关标示的详细信息。

Performance indicators绩效指标。一些反映资源利用率和性能的计数器和仪表。详细可参见“6.1.7 绩效指标”章节。

**6.1.3连接状态**

可以通过查看/proc/drbd或者是使用命令drbdadm cstate 查看资源的连接状态：

drbdadm cstate <resource>

Connected

一个资源可能有以下连接状态之一：

StandAlone独立的：网络配置不可用。资源还没有被连接或者是被管理断开（使用drbdadm disconnect命令），或者是由于出现认证失败或者是裂脑的情况。

Disconnecting断开：断开只是临时状态，下一个状态将是StandAlone独立的。

Unconnected悬空：是尝试连接前的临时状态，可能的下一个状态为WFconnection和WFReportParams。

Timeout超时：与对等节点连接超时，也是临时状态，下一个状态为Unconected悬空。

BrokerPipe：与对等节点连接丢失，也是临时状态，下一个状态为Unconected悬空。

NetworkFailure：与对等节点失去连接后的临时状态，下一个状态为Unconected悬空。

ProtocolError. 与对等节点失去连接后的临时状态，下一个状态为Unconected悬空。

TearDown拆解：临时状态，对等节点连接关闭，下一个状态为Unconected悬空。

WFConnection.等待和对等节点建立网络连接。

WFReportParams：已经建立TCP连接，本节点等待从对等节点传来的第一个网络包。

Connected连接：Drbd已经建立连接，数据镜像现在可用，节点处于正常状态。

StartingSyncS：完全同步，有管理员发起的刚刚开始同步。未来可能的状态为SyncSource或PausedSyncS。

StartingSyncT：完全同步，有管理员发起的刚刚开始同步，下一状态为WFSyncUUID。

WFBitMapS：部分同步刚刚开始，下一步可能的状态：SyncSource或PausedSyncS。

WFBitMapT：部分同步刚刚开始，下一步可能的状态：WFSyncUUID。

WFSyncUUID：同步即将开始，下一步可能的状态：SyncTarget或PausedSyncT。

SyncSource：以本节点为同步源的同步正在进行。

SyncTarget：以本节点为同步目标的同步正在进行。

PausedSyncS：以本地节点是一个持续同步的源，但是目前同步已经暂停。可能是因为另外一个同步正在进行或者是使用命令drbdadm pause-sync暂停了同步。

PausedSyncT：以本地节点为持续的同步目标，但是目前同步已经暂停，这可能是因为另外一个同步正在进行或者是使用命令drbdadm pause-sync暂停了同步。

VerifyS：以本地节点为验证源的线上设备验证正在执行。

VerifyT：以本地节点为目标源的线上设备验证正在执行。

**6.1.4资源角色**

可以通过查看/proc/drbd或者是使用命令drbdadm role 查看资源角色：

drbdadm role <resource>

Primary/Secondary

本地节点一般显示在前，对等节点资源显示在后。

资源的角色一般为如下之一：

Primary主：资源目前为主，并且可能正在被读取或者写入。若不是双主模式被激活，这种角色只可能出现在两节点中的一个上。

Secondary次：资源目前为次。正常接收对等节点的更新（除非运行在断开模式下才不是），但是它既不能被读取也不能被写入。这种角色只能是两节点中的一个。

Unknown未知：资源角色目前未知。本地资源不会出现这种状态，只有对等节点在断开模式下才会出现在这种情况。

**6.1.5 磁盘状态**

可以通过查看/proc/drbd或者是使用命令drbdadm role 查看资源的磁盘状态：

drbdadm dstate <resource>

UpToDate/UpToDate

本地节点一般显示在前，对等节点资源显示在后。

本地和对等节点的磁盘状态都有可能是一下状态之一：

Diskless无盘：本地没有块设备分配给DRBD使用，这就意味着没有可用的后背设备，或者使用drbdadm 命令手工分离或者是底层的I/O错误导致自动分离。

Attaching：读取元数据时候的瞬间状态。

Failed失败：本地块设备报告I/O错误的下一个状态。其下一个状态为Diskless无盘。

Negotiating：在已经连接的DRBD设备进行Attach读取元数据前的瞬间状态。

Inconsistent：数据是不一致的，在两个节点上（初始的完全同步前）这种状态出现后立即创建一个新的资源。此外，在同步期间（同步目标）在一个节点上出现这种状态。

Outdated：数据资源是一致的，但是已经过时。

DUnknown：当对等节点网络连接不可用时出现这种状态。

Consistent：一个没有连接的节点数据一致。当建立连接时，它决定数据是UpToDate 或者是Outdated。

UpToDate：一致的最新的数据状态，这个状态是正常的状态。

**6.1.6I / O的状态标志**

I/O状态标志包含在/proc/drbd中的有关I/O操作的当前资源的信息。I/O状态表示一共六种，具体信息如下：

1.     I/O suspension，r表示运行I/O，s表示suspended I/O，正常情况下为r

2.     Serial resynchronization 串行同步：当资源等待同步时却推迟了重新同步，这个标志变成A，正常情况为 –

3.     Peer-initiated sync suspension 对等节点发起的同步暂停：当资源正在等待重新同步时，对等节点因为一些原因发起暂停同步。这个标志变成p。正常为 –

4.     Locally initiated sync suspension 本地发起的同步暂停：当资源等待重新同步时，在本地节点上用户发起同步暂停，这个标志变成u。正常为 –

5.     Locally blocked I/O 。正常为 -。可能会是一下标志：

d：如因为一个短暂的磁盘状态导致的drbd内部出现I/O阻塞

b：备用设备的I/O阻塞

n：网络socket出现阻塞

a：设备的I/O阻塞和网络阻塞的组合

6.     Activity Log update suspension 活动日志更新暂停：当活动日志更新暂停，这个标志变成s。正常为 –

**6.1.7绩效指标**

/proc/drbd第二行包含每个资源的计数器和仪表状况的信息：

NS（network send 网络发送）：net数据以kibyte通过网络连接发送到对等节点

NR（network receive 网络接收）：通过网络连接以kibyte接收net数据

DW（disk write 磁盘写入）：net数据以kibyte写入本地磁盘

DR（disk read 磁盘读取）：net数据以kibyte读取本地磁盘数据

AL（activity log 活动日志）：活动日志区的元数据更新

BM（bit map 位图）：位图区元数据更新

LO（local count 本地计数）：DRBD请求的开放本地I/O子系统的数量

PE（pengding 待定）：发送到对等节点但是尚未节点对方回答的请求的数目

UA（unacknowledged 未确认）：通过网络连接接收到对等节点的请求，但是尚未得到回复

AP（alllication pending应用程序挂起）：数据块I/O请求转发到DRBD，但是DRBD尚未回答

EP（epochs）：一定数量的epoch对象，通常为1，使用阻碍或者是没有排序写的方法时可能增加I/O负载。

WO（write order 写入顺序）目前使用的写入顺序的方法：b(barrier障碍)、f（flush 刷新）、d(drain 漏)或者是n（none无）

OOS（out of sync）：以kububyte同步当前存储

**6.2启用和禁用资源**

**6.2.1启用资源**

通常情况下，所有配置的DRBD资源自动启用。

**在其自由裁量权的范围内管理集群资源或者是根据集群的配置管理**

**在系统启动时使用/etc/init.d/drbd 脚本**

不管什么理由，如果你要手动启用资源，你可以通过如下名下进行操作：

drbdadm up <resource>

当然你也可使用all这个关键字，而不是特定的资源名称来启动/etc/drbd.conf定义的所有资源。

**6.2.2禁用资源**

可以通过命令暂时禁用特定的资源：

drbdadm down <resource>

这里同样可以使用all这个关键字暂时停用所有在/etc/drbd.conf配置的所有资源。

**6.3重新配置资源**

DRBD运行在操作的过程中重新配置资源。

**要保证对资源的任何必要的修改都配置在/etc/drbd.conf中**

**在两个节点中同步/etc/drbd.conf**

**在量节点上都需要使用drbdadm adjust <resource>**

Drbdadm adjust 后需要使用drbdsetup 配置进行必要的调整。当然你可以通过运行drbdadm -d(dry-run)选项检查drbdsetup的调用。

|  |
| --- |
| **备忘**  当调整/etc/drbd.conf中常见的部分，可以使用命令drbdadm调整所有运行中的资源。 |

**6.4升级和降级资源**

可以通过手工发出如下命令更改资源的角色，或者从次切换为主或者是从主到次：

drbdadm primary <resource>

drbdadm secondary <resource>

在单主模式下（DRBD默认），两个节点同时处于连接状态下，任何一个节点都可以在特定的时间内变成主。当在一个节点上运行 drbdadm primary <resource>后再另外一个节点又运行drbdadm primary <resource>，这样就会出现错误。

但是在双主模式下，可以将两个节点上的资源的角色都可以是主。

**6.5启用双主模式**

双主模式下允许一个资源同事承担两个节点上的主。这样做可能是出于永久或者临时的基础上考虑。

|  |
| --- |
| **备忘**  双主模式要求资源配置为同步复制（协议C） |

**6.5.1永久双主模式**

如若启用双主模式，则需要在资源配置中指定 allow-two-primaries 为yes，如下：

resource <resource>

  net {

    protocol C;

    allow-two-primaries yes;

  }

  ...

}

配置之后不要忘记在两节点间同步配置。在两节点上运行drbdadm adjust <resource>。

之后就可以在两个节点上同时运行 drbdadm primary <resource>。

**6.5.2临时双主模式**

如若在一个单主模式的配置下临时调整启用双主模式，可以使用以下命令：

drbdadm net-options --protocol=C --allow-two-primaries <resource>

要结束临时双主模式，同样可以运行上面类似的命令，不过参数为 --allow-two-primaries=no（如果需要可以跟上所需要的复制协议）

**6.5.3开机自动启用**

当配置资源为双主模式时，也可以配置为在系统开机时自动启动drbd并切换为主模式。

resource <resource>

  startup {

    become-primary-on both;

  }

  ...

}

可利用系统脚本/etc/init.d/drbd作为系统启动和推广相关资源。

|  |
| --- |
| **备忘**  在Pacemaker管理的drbd配置中，成为主的方法不要求和不推荐。Pacemaker配置中，资源的升降级应该有集群进行管理。 |

**6.6在线设备验证**

**6.6.1启用在线设备验证**

默认情况下在线设备验证是未启用的，如果要启用它，可以在配置文件/etc/drbd.conf资源配置部分添加：

resource <resource>

  net {

    verify-alg <algorithm>;

  }

  ...

}

<algorithm>算法，系统内核配置系统密码api支持任意的信息加密算法。通常情况下，你至少可以从sha1、md5和crc32c中选择其一。

如果对现有资源进行改更改，同样需要同步drbd.conf到对等节点，需要在两个节点上运行drbda adjust <resource>。

**6.6.2调用在线验证**

在启用在线验证后，可以通过如下命令启动验证：

drbdadm verify <resource>

这样做之后drbd就开始对<resource>进行在线设备验证，如果探测到有块未同步，将会对这些块进行标记，并向内核日志中记录一条信息。此时任何应用程序都可正常操作，包括切换资源的角色。

如果在验证运行时out-of-sync块被发现，可能需要在验证完毕之后使用如下命令通信同步：

drbdadm disconnect <resource>

drbdadm connect <resource>:

**6.6.3自动在线验证**

大多数用户为轻松完成工作，都希望自动进行在线设备验证。可以创建一个名为/etc/dron.d/drbd-verify的文件，输入如下内容：

42 0 \* \* 0    root    /sbin/drbdadm verify <resource>

这样在每星期日的午夜的00:42都会进行设备的在线验证。

如果要对所有的资源启用在线设备验证（如通过在/etc/drbd.conf文件的common部分添加verify-alg <algorithm>），也可以使用如下进行调用：

42 0 \* \* 0    root    /sbin/drbdadm verify all

**6.7配置同步率**

正常情况下，后台同步需要保证（这会导致同步目标出现暂时的不一致）尽可能快的完成，但是后台同步带宽不能占用所有的带宽，不然会影响前端复制，影响应用程序的性能。因此，也就必须根据各自的硬件环境，配置同步的带宽。可以是永久的也可能是临时的。

|  |
| --- |
| **重要**  **设置的同辈速率比次节点的最大吞吐量大还大的话，会变得无意义。次节点的写速度必会比它的I/O子系统还快，因此它是一个正在进行同步的同步目标设备。** |

基于同样的理由，同步速率设置的超过网络的最大可用带宽也是没有任何意义的。

**6.7.1固定的同步速率配置**

一个资源用于后台同步的最大带宽由资源的速率选项控制。它必须包含在资源配置文件/etc/drbd.conf的磁盘部分：

resource <resource>

  disk {

    sync-rate 40M;

    ...

  }

  ...

}

请注意，同步的速率是bytes字节，而不是bits/s。

|  |
| --- |
| **提示**  **根据经验同步速率比较合理的是可用带宽的30%。这样加入一个I/O子系统能支持180MB/s的吞吐量，而千兆网络可支持110MB/s（网络带宽会成为瓶颈），可以计算出：** |

图6.1。以网络带宽110MB/ s为例，同步速率：

这样就得出同步速率的推荐值为33M。

相反，如果I/O子系统的支持80MB/s的吞吐量，千兆以网络环境（I/O子系统是瓶颈）的最大吞吐量应该为：

图6.2。以网络带宽80MB/ s为例，同步速率：

这样就得出同步速率的推荐值为24M。

**6.7.2临时的同步速率配置**

有时候需要暂时调整同步的速率。例如，集群节点进行预定的维护后希望加快后台重新同步，或者是当应用程序非常频繁的进行写的时候，希望降低重新同步的速率以保证比较大的带宽用于复制。

例如，千兆以太网将大部分带宽用于重新同步，可以使用如下命令：

drbdadm disk-options --resync-rate=110M <resource>

只需要在一个节点上发出以上命令。

要恢复这个临时设置，并启用/etc/drbd.conf中配置的同步速率，可以使用如下命令：

drbdadm adjust <resource>

**6.7.3可变同步率配置**

当多个DRBD资源共用同一个复制/同步的网络时，固定的同步速率可能就不是很合适。在这种情况下，可以配置可变的同步速率。DRBD使用一个自动控制回路的算法来确定和调整同步速率。该算法可以保证始终有足够的带宽用于前台的复制，从而大大的减小了后台同步I/O对前台的影响。

可变同步速率的优化配置很大程度上取决于可用的网络带宽，应用I/O模型和链路的拥塞状况。理想的设置还取决于是不是应用了DRBD代理。在优化DRBD的配置方面比较明智的是请教专业的DRBD人员。下面提供一个配置的样例（假定结合了drbd代理）：

resource <resource> {

  net {

    c-plan-ahead 200;

    c-max-rate 10M;

    c-fill-target 15M;

  }

}

|  |
| --- |
| **提示**  **c-fill-target比较好的起始值为BDP\*3，其中BDP为复制链接产生的带宽延迟** |

**6.8配置基于校验的同步**

默认情况下基于校验的同步是未启用的。如果要启用的话可以在配置文件/etc/drbd.conf中配置：

resource <resource>

  net {

    csums-alg <algorithm>;

  }

  ...

}

<algorithm>算法，系统内核配置系统密码api支持任意的信息加密算法。通常情况下，你至少可以从sha1、md5和crc32c中选择其一。

如果对现有资源进行改更改，同样需要同步drbd.conf到对等节点，需要在两个节点上运行drbda adjust <resource>。

**6.9配置拥堵策略和暂停复制**

在复制带宽高度可变的环境下（广域网下的典型），有可能复制就会发生拥堵。使用默认的配置是不可取的，因为这样会导致主节点上I/O阻塞。

相反，如果在这种情况下暂停DRBD的复制，会导致主节点上的数据在此节点上设置为*pull ahead*。在这种模式下，drbd是始终开启着复制通道的-它并没有置换到disconnected断开模式，不过在没有足够的带宽可用前，它并不进行复制操作。

下面的一个drbd proxy的配置样例：

resource <resource> {

  net {

    on-congestion pull-ahead;

    congestion-fill 2G;

    congestion-extents 2000;

    ...

  }

  ...

}

通常同时设置congestion-fill、congestion-extents和pull-ahead选项一起工作比较好。

congestion-fill比较合理的值是90%。

**当通过drbd proxy复制时，drbd proxy 缓冲内存。**

**当不设置drbd proxy 时，通过tcp网络发送缓冲。**

Congestion-extents比较合理的值为设置的al-extents值得90%。

**6.10配置I/O错误处理策略**

DRBD可通过在资源的磁盘配置文件/etc/drbd.conf中配置on-io-error选项处理底层的I/O错误：

resource <resource> {

  disk {

    on-io-error <strategy>;

    ...

  }

  ...

}

如果想对所有的资源做一个全局性的I/O错误处理策略，也可以在common部分设置该选项。

<strategy>策略可能有如下选项之一：

1、  detach分离：这是默认和推荐的选项。如果在节点上发生底层的磁盘I/O错误，它会将设备运行在diskless 无盘模式下。

2、  pass\_on：drbd会将I/O错误报告到上层。在主节点上，它会将其报告给挂载的文件系统，但是在此节点上就往往忽略（因此此节点上没有可以报告的上层）

3、  -local-io-error调用本地磁盘I/O处理程序中定义的命令。这就需要有相应有让local-io-error调用的资源处理程序处理错误的命令。这就给管理员留有足够自由的权力使用命令或者是脚本调用local-io-error处理I/O错误。

|  |
| --- |
| **备忘**  在早起的DRBD版本中（8.0之前）其实包含另外一个选项，panic，它是在本地出现I/O错误时强行将发生I/O错误的节点从集群中移除。虽然该选项不再可用，但是通过local-io-error/+ call-local-io-error+接口可实现与之相同的功能。当你对足够了解这种行为的影响时你可以应用这样的该功能。 |

你可以按照如下的操作重新配置正在运行资源的I/O错误处理策略：

**编辑资源配置文件/etc/drbd.d/<resource>.res**

**将配置文件拷贝到对等节点**

**在两个节点上运行drbdadm adjust <resource>**

**6.11配置复制传输完整性验证**

默认情况下资源的复制传输完整性验证是不开启的。可以在/etc/drbd.conf文件中添加如下内容启用复制传输完整性验证：

resource <resource>

  net {

    data-integrity-alg <algorithm>;

  }

  ...

}

<algorithm>算法，系统内核配置系统密码api支持任意的信息加密算法。通常情况下，你至少可以从sha1、md5和crc32c中选择其一。

如果对现有资源进行改更改，同样需要同步drbd.conf到对等节点，需要在两个节点上运行drbda adjust <resource>。

**6.12重新调整资源**

**6.12.1在线增长**

如果后端块设备支持（在线）操作，那也就可能支持在调整时在线调整DRBD设备的大小。如要这样做，必须满足两个条件：

1.     受影响的后端设备必须是一个逻辑卷管理子系统，如LVM。

2.     资源目前必须出在连接状态。

如果后端块设备在两个节点上都支持增长，就需要保证仅有一个节点处于主状态。然后在一个节点上输入：

drbdadm resize <resource>

新的部分将触发同步。同步从主节点到次节点上。

**6.12.2离线增长**

当两个节点的后端块设备增长时，DRBD是不活动，且drbd利用外部元数据，那么新的大小会自动识别。任何管理的干预都是不必要的。Drbd设备的新大小激活，并建立网络连接。

如果drbd资源配置的是使用内部元数据，那么这个元数据必须在增长的设备变成可用前移动进去。可以采用以下步骤完成操作：

|  |
| --- |
| **警告**  这是一个超级进程，你可以自己觉得如何使用。 |

**取消配置的DRBD资源**

drbdadm down <resource>

**保存萎缩之前的元数据到文本**

drbdadm dump-md <resource> > /tmp/metadata

必须在两个节点上都这样操作，为每个节点都单独的转储文件。千万不要在一个节点导出数据然后拷贝到其他的节点上，这样会导致drbd不能工作。

**两个节点上的后端块设备都支持增长**

**可在两个节点上，根据文件/tmp/metadata调整大小信息（la-size-sect）。请注意参数la-size-sect必须在sectors中指定。**

**重新初始化元数据区域：**

drbdadm create-md <resource>

**在两个节点上重新导入正确的元数据：**

drbdmeta\_cmd=$(drbdadm -d dump-md test-disk)

${drbdmeta\_cmd/dump-md/restore-md} /tmp/metadata

Valid meta-data in place, overwrite? [need to type 'yes' to confirm]

yes

Successfully restored meta data

|  |
| --- |
| **备忘**  这个例子使用shell变量代替。它可能或不能在其他的shell中工作。检查环境变量看是不是没有正确使用。 |

**重新启用drbd资源：**

drbdadm up <resource>

**在一个节点上提升drbd的资源：**

drbdadm primary <resource>

**最后，增长的文件系统填补扩展的drbd的设备的大小。**

**6.12.3在线收缩**

在收缩drbd设备前，必须先缩小drbd的上层，例如，通常的文件系统系统。由于drbd不能获取到文件系统实际使用多少空间，为了不丢失数据操作一定要小心。

|  |
| --- |
| **备忘**  能不能在线收缩取决于正在使用的文件系统。大多数的文件系统是不支持在线收缩的。如xfs根本不支持收缩。 |

当使用内部元数据时一定要考虑元数据使用的磁盘空间。提供给drbdadm resize的大小是文件系统的大小。在使用内部元数据的情况下，drbd所需求的大小要高（详见17.1.3 计算元数据大小  一节）。

要在线收缩drbd，可以在收缩文件系统之后使用如下命令进行操作：

drbdadm resize --size=<new-size> <resource>

通常可以使用乘数后缀为<new-size>(K/M/G等)，在收缩drbd之后，还可以收缩包含的块设备（如果支持收缩的话）。

**6.12.4离线收缩**

如果收缩不活动的drbd块设备，在尝试连接时，drbd就会拒绝连接到块设备，因为目前它太小（使用外部元数据的情况下），或者它不能找到内部元数据（使用内部元数据的情况下）。为解决该问题，使用该程序（如果不能使用在线收缩）：

|  |
| --- |
| **警告**  这是一个超级进程，你可以自己觉得如何使用。 |

**要在一个节点上收缩文件系统，依然要配置drbd**

**取消配置的drbd资源：**

drbdadm down <resource>

**保存萎缩之前的元数据到文本**

drbdadm dump-md <resource> >+ /tmp/metadata

必须在两个节点上都这样操作，为每个节点都单独的转储文件。千万不要在一个节点导出数据然后拷贝到其他的节点上，这样会导致drbd不能工作。

**在两个节点上收缩后端块设备**

**可在两个节点上，根据文件/tmp/metadata调整大小信息（la-size-sect）。请注意参数la-size-sect必须在sectors中指定。**

**只有当你使用内部元数据（此时可能由于收缩进行丢失），重新初始化元数据区域：**

drbdadm create-md <resource>

**在两个节点上重新导入正确的元数据：**

drbdmeta\_cmd=$(drbdadm -d dump-md test-disk)

${drbdmeta\_cmd/dump-md/restore-md} /tmp/metadata

Valid meta-data in place, overwrite? [need to type 'yes' to confirm]

yes

Successfully restored meta data

|  |
| --- |
| **备忘**  这个例子使用shell变量代替。它可能或不能在其他的shell中工作。检查环境变量看是不是没有正确使用。 |

**重新启用drbd资源：**

drbdadm up <resource>

**6.13禁用后端设备刷新**

|  |
| --- |
| **注意**  当有耐性的drbd使用battery-backed高速写缓存（BBWC）时，应该只禁用设备的刷新。当battery池耗尽时大多数的存储控制器允许自动禁用写缓存，当battery死时切换到写模式。强烈建议使用该功能。 |

当运行无BBWC或者是BBWC池耗尽时，禁用DRBD缓存很可能会导致数据丢失，因此不应该尝试这样做。

DRBD允许分别启用和禁用后端设备刷新对复制数据集和DRBD拥有自身的元数据。这写选项默认都是开启的。如果要禁用一个（或者是两个），就需要设置drbd配置文件中的/etc/drbd.conf中的disk部分。

要禁用复制数据集的磁盘刷新，就需要配置如下：

resource <resource>

  disk {

    disk-flushes no;

    ...

  }

  ...

}

要禁用drbd元数据的磁盘刷新，需要配置如下：

resource <resource>

  disk {

    md-flushes no;

    ...

  }

  ...

}

在核对资源配置后（同样需要同步/etc/drbd.conf到两个节点间），可以在两个节点运行如下命令：

drbdadm adjust <resource>

**6.14配置裂脑行为**

**6.14.1裂脑通知**

如果配置DRBD会调用裂脑处理程序，裂脑发生时就会被探测到。要配置这个程序，需要对资源进行如下配置：

resource <resource>

  handlers {

    split-brain <handler>;

    ...

  }

  ...

}

<handler>可能是目前系统中一个可执行的文件。

Drbd包含一个裂脑处理程序脚本/usr/lob/drbd/notify-split-brain.sh。它这是通过电子邮件的方式发送到指定的地址。要配合程序发送信息到root@localhost（这假设是设置的系统管理员的邮件地址），配置如下：

resource <resource>

  handlers {

    split-brain "/usr/lib/drbd/notify-split-brain.sh root";

    ...

  }

  ...

}

当配置已经在资源上进行修改（同步到两个节点上），就不需要添加其他的处理就可以启动处理程序。Drbd会在下一次出现裂脑时候就简单的调用该处理程序。

**6.14.2 裂脑自动修复策略**

为了能够启用和配置drbd的自动裂脑恢复策略，就必须理解drbd提供的几种配置选项。Drbd提供基于同时几个几点处于主角色探测的裂脑恢复程序。为此，drbd资源的net配置部分有以下关键字：

after-sb-0pri：裂脑已经被探测到，但是现在没有节点处于主角色，对于这个选项，drbd有以下关键字：

**disconnect:不需要自动恢复，仅仅是调用裂脑处理程序的脚本（如果配置了），断开连接并出在断开模式。**

**discard-younger-primary:放弃和回滚最后成为主的上面所做的修改。**

**discard-least-changes:放弃和回滚，变动比较少的主机上的修改。**

**discard-zero-changes:如果任何节点都没有发生任何变化，仅仅申请在一个节点上做出继续修改即可。**

after-sb-1pri：裂脑已经被探测到，现有有一个节点处于主角色，对于这个选项，drbd有以下关键字：

**disconnect:和after-sb-0pri一样，调用裂脑处理程序的脚本（如果配置了），断开连接并出在断开模式。**

**consensus:和after-sb-0pri中同样的修复策略。如果利用这些策略裂脑危害能选择，那就能自动解决。否则，同样断开指定的动作。**

**call-pri-lost-after-sb:和after-sb-0pri中同样的修复策略。如果利用这些策略裂脑危害能选择，就在受危害的节点上调用pri-lost-after-sb程序。这个程序必须确认在handlers中配置，并考虑到从集群中移除该节点。**

**discard-secondary:不管哪个主机只要处于次角色，都是裂脑的危害者。**

after-sb-2pri：在两个节点都处于主角色时，裂脑被发现。次选项使用和after-sb-1pri同样的关键字，丢弃次节点并达成共识。

|  |
| --- |
| **备忘**  DRBD的这三个选项，往往被忽略，因为很少用到。可以参见drbd.cong(5)查看裂脑恢复的详细关键字，这里不再细说。 |

例如，在双主模式下，使用GFS或者OCFS2文件系统作为资源的块设备，可以使用如下方式定义恢复策略：

resource <resource> {

  handlers {

    split-brain "/usr/lib/drbd/notify-split-brain.sh root"

    ...

  }

  net {

    after-sb-0pri discard-zero-changes;

    after-sb-1pri discard-secondary;

    after-sb-2pri disconnect;

    ...

  }

  ...

}

**6.15创建三节点设置**

三节点的设置涉及DRBD设备之上另一个堆叠。

**6.15.1设备堆叠的注意事项**

这种类型的设置需要注意一下事项：

**堆叠的设备是活动的设备。假若已经配置了drbd设备/dev/drbd0，并且在堆叠设备上配置的为/dev/drbd10，那么/dev/drbd10这就是挂载和使用的设备。**

**设备的元数据将被保持两次，分别为底层的drbd设备和堆叠的设备上。在堆叠的设备上，必须使用内部元数据，这就意味着堆叠的设备比不堆叠的设备有效可用的存储区域要小。**

**如要上层的堆叠设备运行，那么底层的设备必须是主。**

**为了保证能够同步到备份节点，堆叠设备必须是活动并且是主角色。**

**6.15.2配置堆叠的资源**

在下面的例子中，节点分别命名为alice、bob和charlie，而alice和bob形成一个两节点的集群，而charlie则是备份的节点。

resource r0 {

  net {

    protocol C;

  }

  on alice {

    device     /dev/drbd0;

    disk       /dev/sda6;

    address    10.0.0.1:7788;

    meta-disk internal;

  }

  on bob {

    device    /dev/drbd0;

    disk      /dev/sda6;

    address   10.0.0.2:7788;

    meta-disk internal;

  }

}

resource r0-U {

  net {

    protocol A;

  }

  stacked-on-top-of r0 {

    device     /dev/drbd10;

    address    192.168.42.1:7788;

  }

  on charlie {

    device     /dev/drbd10;

    disk       /dev/hda6;

    address    192.168.42.2:7788; # Public IP of the backup node

    meta-disk  internal;

  }

}

在有三个节点的情况下，配置文件drbd.conf必须在每一个集群的节点中都有。注意以下关键字在不堆叠的资源配置中是没有的：

stacked-on-top-of：该选项表示drbd所使用的资源为一个堆叠的资源。它替换普通的资源配置的部分。在底层的资源中不能使用stacked-on-top-of选项。

|  |
| --- |
| **备忘**  DRBD的这三个选项，往往被忽略，因为很少用到。可以参见drbd.cong(5)查看裂脑恢复的详细关键字，这里不再细说。 |

对堆叠的资源来说Protocol A不是必须的。你可以根据需求，选择任何drbd的复制协议。

**6.15.3启用堆叠资源**

为了启用堆叠资源，首先必须先启用底层的资源并推广：

drbdadm up r0

drbdadm primary r0

而对不堆叠的资源，则在必须堆叠的资源上创建drbd元数据。可是使用如下命令：

drbdadm create-md --stacked r0-U

然后你就可以启用堆叠的资源了：

drbdadm up --stacked r0-U

drbdadm primary --stacked r0-U

在此之后，你可以在备用节点上启动资源，开启三节点复制：

drbdadm create-md r0-U

drbdadm up r0-U

为了自动化管理堆叠的资源，必须在集群管理配置中整合堆叠的资源。参见第8.4节“Pacemaker集群中使用堆叠的DRBD资源”了解在Pacemaker集群中管理集群的一些详细信息。

**6.16使用DRBD proxy**

**6.16.1DRBD proxy部署注意事项**

DRBD proxy进程即可直接放在DRBD自身的服务器上，也可以放在其他的单独的专用服务器上。DRBD proxy实例可作为多个drbd设备分布在多个节点上的proxy代理。

Drbd proxy对drbd来说是透明的。通常情况下会有大量的数据报进行传输，这样就有相当大的活动日志。这就可能导致时间比较长的同步在主节点崩溃后重新运行，因此建议启用drbd的csums-alg设置。

**6.16.2安装**

请联系linbit销售商获取drbd proxy。除非有特殊需求，要不就请使用最新的drbd proxy。

在drbian和debian-based系统上安装drbd proxy，使用dpkg工具操作如下(使用的drbd proxy的版本要和目标架构的版本相匹配)：

dpkg -i drbd-proxy\_1.0.16\_i386.deb

在基于rpm的系统（如sles和rhel）上安装drbd proxy，使用rpm工程操作如下(使用的drbd proxy的版本要和目标架构的版本相匹配)：

rpm -i drbd-proxy-1.0.16-1.i386.rpm

要配置drbd proxy还需要安装drbd管理工具drbdadm。

以二进制文件安装drbd proxy通常会安装到/etc/init,d的init脚本中。请使用init脚本start/stop drbd proxy服务，它相应需要使用drbdadm工具配置drbd proxy。

**6.16.3License 文件**

当从linbit获取一个license时，drbd proxy的运行也是需要license文件的。这个文件名为drbd-proxy.license，必须在目标机器上拷贝到/etc目录下：

cp drbd-proxy.license /etc

**6.16.4配置**

Drbd proxy在drbd的主被配置文件中配置。它是在主机部分中的proxy和additional proxy的additional选项部分进行配置。

下面为在drbd节点上直接配置运行drbd proxy的例子：

resource r0 {

        net {

          protocol A;

        }

        device     minor 0;

        disk       /dev/sdb1;

        meta-disk  /dev/sdb2;

        proxy {

              compression on;

              memlimit 100M;

        }

        on alice {

                address 127.0.0.1:7789;

                proxy on alice {

                        inside 127.0.0.1:7788;

                        outside 192.168.23.1:7788;

                }

        }

        on bob {

                address 127.0.0.1:7789;

                proxy on bob {

                        inside 127.0.0.1:7788;

                        outside 192.168.23.2:7788;

                }

        }

}

Insert ip地址为drbd和drbd proxy之间沟通使用，而outside ip地址为drbd proxy之间沟通使用。

**6.16.5控制DRBD proxy**

Drbdadm提供proxy-up和proxy-down子命令配置或删除连接到本地drbd资源的drbd proxy。可使用/etc/init.d/drbdproxy命令进行启动和停止的操作。

Drbd proxy有一个焦作drbd-proxy-ctl的底层的配置工具。当没有其他的选项时它可在交互模式下调用。帮助信息命令显示如下：

add connection <name> <ip-listen1>:<port> <ip-connect1>:<port>

   <ip-listen2>:<port> <ip-connect2>:<port>

   Creates a communication path between two DRBD instances.

set memlimit <name> <memlimit-in-bytes>

   Sets memlimit for connection <name>

del connection <name>

   Deletes communication path named name.

show

   Shows currently configured communication paths.

show memusage

   Shows memory usage of each connection.

list [h]subconnections

   Shows currently established individual connections

   together with some stats. With h outputs bytes in human

   readable format.

list [h]connections

   Shows currently configured connections and their states

   With h outputs bytes in human readable format.

list details

   Shows currently established individual connections with

   counters for each DRBD packet type.

quit

   Exits the client program (closes control connection).

shutdown

   Shuts down the drbd-proxy program. Attention: this

   unconditionally terminates any DRBD connections running.

**6.16.6故障排除**

Drbd proxy日志通过syslog使用LOG\_DAEMON功能。通常情况下，可在/var/log/daemon.log查看drbd proxy信息。

例如，如果drbd 连接失败，因为本节点不能连接到领导的一端，就会记录一些类似拒绝连接的信息。在这种情况下酒需要检查DRBD是不是云翔在两个节点上（不是standalone模式），并且drbd proxy在运行。同时也需要自己检查配置。

**7故障排除和错误**

目录

[7.1磁盘故障的处理](http://write.blog.csdn.net/postedit/6460814#_7.1磁盘故障的处理)

[7.1.1从磁盘中手工分离出DRBD](http://write.blog.csdn.net/postedit/6460814#_7.1.1从磁盘中手工分离出DRBD)

[7.1.2自动分离I / O错误](http://write.blog.csdn.net/postedit/6460814#_7.1.2自动分离I_/_O错误)

[7.1.3使用内部元数据时更换故障磁盘](http://write.blog.csdn.net/postedit/6460814#_7.1.3使用内部元数据时更换故障磁盘)

[7.1.4使用外部元数据时更换故障磁盘](http://write.blog.csdn.net/postedit/6460814#_7.1.4使用外部元数据时更换故障磁盘)

[7.2处理节点故障](http://write.blog.csdn.net/postedit/6460814#_7.2处理节点故障)

[7.2.1处理次节点临时故障](http://write.blog.csdn.net/postedit/6460814#_7.2.1处理次节点临时故障)

[7.2.2处理主节点临时故障](http://write.blog.csdn.net/postedit/6460814#_7.2.2处理主节点临时故障)

[7.2.3节点永久故障的处理](http://write.blog.csdn.net/postedit/6460814#_7.2.3节点永久故障的处理)

[7.3手工进行裂脑修复](http://write.blog.csdn.net/postedit/6460814#_7.3手工进行裂脑修复)

本章介绍在硬件或系统故障的情况下执行的任务

**7.1磁盘故障的处理**

如何处理磁盘故障取决于drbd配置的磁盘I/O错误处理策略（参阅2.11节“磁盘错误处理策略”），和元数据类型的配置（参阅17.1节“DRBD元数据”）。

|  |
| --- |
| 备注  在大多数情况下，这里所描述的步骤仅仅使用DRBD直接运行在磁盘上的时候。一般情况下，对DRBD运行在顶层时不适用。 |

**MD软raid设置（在这种情况下使用mdadm管理磁盘的更换），**

**设备映射raid（使用dmraid）**

**硬raid设备（可根据供应商的指示处理磁盘故障）**

**一些非标准设备映射的虚拟块设备（参阅设备映射的文档）**

**7.1.1从磁盘中手工分离出DRBD**

如果drbd配置通过I/O错误（不推荐），就必须先分离drbd的资源，即，分离出后端存储：

drbdadm detach <resource>

可以运行drbdadm dstate命令，查看资源是不是运行在diskless无盘模式：

drbdadm dstate <resource>

Diskless/UpToDate

如果主节点发生磁盘故障，可以结果这一步完成切换操作。

**7.1.2自动分离I / O错误**

如果drbd配置为自动分离I/O错误（推荐选项），DRBD就可以从后端存储上自动分离资源，而无需人工干预。你可以通过drbdadm dstate命令核对资源是不是运行在diskless无盘模式下。

**7.1.3使用内部元数据时更换故障磁盘**

如果使用内部元数据，就足以将drbd设备绑定到新的磁盘上。如果新的磁盘名被其他的linxu设备或者是故障磁盘占用，就需要修改drbd的配置文件进行修改。

这个过程包括新建元数据集，然后从新连接资源：

drbdadm create-md <resource>

v08 Magic number not found

Writing meta data...

initialising activity log

NOT initializing bitmap

New drbd meta data block sucessfully created.

drbdadm attach <resource>

新磁盘的完全同步是瞬间和自动进行的。可以通过查看/proc/drbd检测后台同步的进度信息。

**7.1.4使用外部元数据时更换故障磁盘**

使用外部元数据时，程序基本上是一样的。然而，DRBD不能识别单独的磁盘更换，因此需要一个额外的操作步骤：

drbdadm create-md <resource>

v08 Magic number not found

Writing meta data...

initialising activity log

NOT initializing bitmap

New drbd meta data block sucessfully created.

drbdadm attach <resource>

drbdadm invalidate <resource>

这里drbdadm无效命令触发同步。同样可以通过查看/proc/drbd观察同步的进展情况。

**7.2处理节点故障**

当drbd探测到对等节点宕机（无论是真正的硬件故障还是人工干预），drbd都将其连接状态有Connected变为WFConnection，直到对等节点恢复。Drbd资源会在此时运行disconnected模式下。在disconnected模式下，资源以及和棋相关的块设备都是可用的，并可以在必要时进行升降级，但是并不能将修改复制到对等节点。相反，drbd在块设备上存储的内部信息在disconnected时仍会被修改。

**7.2.1处理次节点临时故障**

如果一个处于次角色的节点资源暂时出现故障（例如因为内存问题导致的需要更换内存），进一步的干预是不需要的—除了明显的必须修复的故障节点并恢复回来。当这种情况发生后，两个节点会重新建立连接并启用，drbd会复制在此期间所有进行的修改到次节点上。

|  |
| --- |
| 重要  由于drbd资源重新同步算法的影响，此时此节点上的资源会出现短暂的不一致inconsistent。在这个短暂的时间内，如果对等节点出现不可用，此时次节点是不能升级到主节点的。因此，在集群不冗余的期间，这个时间为次节点停机的时间加上重新同步所需要的时间。 |

**7.2.2处理主节点临时故障**

从drbd的角度上看，主节点的故障和次节点的故障基本类似。存活的节点发现对等阶段故障，就会将其置为disconnected 模式。Drbd并不升级存活的节点到主，需要集群管理程序重要做。

当失败的节点被修复并回到集群中，在此节点也是如此，因此，无需进行人工干预。再次，drbd并没有改回资源的角色，它是由几圈管理器控制的（如果配置了的话）。

Drbd通过一个特殊的机制保证在主节点出现故障的情况下块设备的一致。详细可以参阅17.3节“活动日志”。

**7.2.3节点永久故障的处理**

如果一个节点遇到不可恢复的问题或者是永久性破坏，需按照以下步骤进行操作：

**将出现故障的硬件替换为与之类似性能和容量的磁盘。**

|  |
| --- |
| 备注  替换也可以替换性能差点，但是不推荐。替换为磁盘容量比较小的是不可能的，这样会导致drbd拒绝连接被替换的节点。 |

**安装基本系统和应用程序。**

**安装drbd并从幸存的节点上拷贝/etc/drbd.conf和所有的/etc/drbd.d**

**按照第五章的步骤配置drbd，但是不进行5.5节的“初始化设备同步”**

此时没有必要手工进行一个完整的设备同步，它会在开始时自动连接到存活的主节点上。

**7.3手工进行裂脑修复**

DRBD检测到裂脑恢复连接并变成可用，和对等节点达成初步的握手。如果drbd检测到两个节点（也可能是两个节点断开时）都是主角色，它就连接关闭复制的连接。这个可在系统日志中发现如下信息：

Split-Brain detected, dropping connection!

当裂脑被探测到时，一个节点将保持始终以StandAlone状态连接资源，另外一个节点也可能处于StandAlone状态（如果两个节点被探测到同时处于裂脑状态），也可能是WFConnection状态（如果对等接在还没有来得及探测到裂脑就down掉的话）。

此时，必须手工干预选择丢失一个节点的修改被丢失（这个节点被称为裂脑受害者），除非配置drbd的裂脑自动修复。这种干预需运行如下命令：

drbdadm secondary <resource>

drbdadm connect --discard-my-data <resource>

在其他节点上（裂脑幸存者），如果它的状态也为StandAlone状态，可输入以下命令：

drbdadm connect <resource>

如果节点已经处于WFConnection状态，可是省略这一步，因为它会自动进行重新连接。

如果裂脑影响的是堆叠的资源，需要使用drbdadm –stacked来代替drbdadm。

当连接时裂脑受害者将立即改变连接状态为SyncTarget，并被主节点覆盖其余节点的修改。

|  |
| --- |
| 备注  裂脑受害者不是一个完整的设备同步，相反，还有可能执行本地修改的回滚操作，而将裂脑幸存者的所做的修改完整的传输到裂脑受害者。 |

当重新完成同步后，就认为裂脑问题已经解决，两个节点的数据再次达成一致，形成一个冗余复制的存储系统。

**第四篇 应用DRBD功能**

**8 DRBD集成Pacemaker集群**

目录

[8 DRBD集成Pacemaker集群](http://write.blog.csdn.net/postedit/6460814#_8_DRBD集成Pacemaker集群)

[8.1 Pacemaker引子](http://write.blog.csdn.net/postedit/6460814#_8.1_Pacemaker引子)

[8.2集群配置添加DRBD-backed服务](http://write.blog.csdn.net/postedit/6460814#_8.2集群配置添加DRBD-backed服务)

[8.3在Pacemaker集群中使用resource-level fencing](http://write.blog.csdn.net/postedit/6460814#_8.3在Pacemaker集群中使用resource-level_fe)

[8.3.1 Resource-level fencing和dopd](http://write.blog.csdn.net/postedit/6460814#_8.3.1_Resource-level_fencing和dopd)

[8.3.2 Resource-level fencing使用集群信息库（CIB）](http://write.blog.csdn.net/postedit/6460814#_8.3.2_Resource-level_fencing使用集群信息库)

[8.4 Pacemaker集群只用堆叠的drbd资源](http://write.blog.csdn.net/postedit/6460814#_8.4_Pacemaker集群只用堆叠的drbd资源)

[8.4.1 添加站点灾难恢复和Pacemaker集群](http://write.blog.csdn.net/postedit/6460814#_8.4.1_添加站点灾难恢复和Pacemaker集群)

[8.4.2使用堆叠的资源，实现4路冗余的Pacemaker集群](http://write.blog.csdn.net/postedit/6460814#_8.4.2使用堆叠的资源，实现4路冗余的Pacemaker集群)

[8.5配置drbd在两个SAN-backed进行复制的Pacemaker集群](http://write.blog.csdn.net/postedit/6460814#_8.5配置drbd在两个SAN-backed进行复制的Pacemake)

[8.5.1 drbd的资源配置](http://write.blog.csdn.net/postedit/6460814#_8.5.1_drbd的资源配置)

[8.5.2 Pacemaker资源的配置](http://write.blog.csdn.net/postedit/6460814#_8.5.2_Pacemaker资源的配置)

[8.5.3 Site的故障转移](http://write.blog.csdn.net/postedit/6460814#_8.5.3_Site的故障转移)

在实际应用中drbd结合Pacemaker集群一起应用是比较常见的。Pacemaker也是DRBD众多应用中比较强大的一个应用。

**8.1 Pacemaker引子**

在linux平台上Pacemaker是一个复杂的、功能丰富并且广泛应用和部署的集群。它配备有丰富的文档集。为了了解这一章，强烈建议阅读以下文件：

**“从头开始，一步一步教你配置高可用性集群” –“Clusters From Scratch, a step-by-step guide to configuring high availability clusters;”**

**CRM CLI（命令行界面）工具，CRM shell手册，Pacemaker的简单直观的命令行界面—“CRM CLI (command line interface) tool, a manual for the CRM shell, a simple and intuitive command line interface bundled with Pacemaker;”**

**Pacemaker配置说明。解释Pacemaker的设计概念和原理的参考文档。**

**8.2集群配置添加DRBD-backed服务**

这部分主要介绍如何启用Pacemaker集群的DRBD-backed服务。

|  |
| --- |
| 备注  如果采用了drbd ocf资源代理，建议推迟drbd的启用、关闭、推广和升降级专门的OCF资源代理。也就是说需要禁用drbd的init脚本（自启动脚本）： |

chkconfig drbd off

ocf：linbit：drbd OCF代理提供主/从功能，允许Pacemaker启动和监控、推广和升降级多个节点的drbd资源。同时也必须了解drbd的 RA断开和分离所有的在Pacemaker管理drbd资源的关闭，也可为另外一个节点启用待机模式。

|  |
| --- |
| 重要  OCF资源和drbd都属于linbit提供商，因此安装为/usr/lib/ocf/resource.d/linbit/drbd。OCF资源代理作为传统的资源代理，提供心跳检测，并安装到 /usr/lib/ocf/resource.d/heartbeat/drbd。传统的OCF RA已经被废弃并不再使用。 |

当Pacemaker CRM集群中使用OCF资源代理时为了启用mysql数据库的drbd-backed配置，为保证服务预先启动提升drbd的资源，必须同时创建必要的资源和Pacemaker 限制。可如下例中使用crm shell：

**Drbd-backed的mysql服务的Pacemaker配置**

crm configure

crm(live)configure# primitive drbd\_mysql ocf:linbit:drbd \

                    params drbd\_resource="mysql" \

                    op monitor interval="15s"

crm(live)configure# ms ms\_drbd\_mysql drbd\_mysql \

                    meta master-max="1" master-node-max="1" \

                         clone-max="2" clone-node-max="1" \

                         notify="true"

crm(live)configure# primitive fs\_mysql ocf:heartbeat:Filesystem \

                    params device="/dev/drbd/by-res/mysql" \

                      directory="/var/lib/mysql" fstype="ext3"

crm(live)configure# primitive ip\_mysql ocf:heartbeat:IPaddr2 \

                    params ip="10.9.42.1" nic="eth0"

crm(live)configure# primitive mysqld lsb:mysqld

crm(live)configure# group mysql fs\_mysql ip\_mysql mysqld

crm(live)configure# colocation mysql\_on\_drbd \

                      inf: mysql ms\_drbd\_mysql:Master

crm(live)configure# order mysql\_after\_drbd \

                      inf: ms\_drbd\_mysql:promote mysql:start

crm(live)configure# commit

crm(live)configure# exit

bye

在此之后，配置应该被启用。Pacemaker现在选择一个节点升级drbd的资源，然后再找个节点上启用drbd-backed资源。

**8.3在Pacemaker集群中使用resource-level fencing**

本节概述为防止Pacemaker从促进drbd的主/备资源时，drbd复制链接被中断的必要采取的步骤。这使Pacemaker开启服务时使用过时的数据，从而导致不必要的“time warp”。

为了保证drbd的resource-level fencing，必须添加以下几行到资源的配置中：

resource <resource> {

  disk {

    fencing resource-only;

    ...

  }

}

由于集群基础设施的使用，也需要对handlers部分进行修改：

**基于心跳的Pacemaker集群可参看“8.3.1 Resource-level fencing和dopd”的配置概述。**

**Corosync和基于心跳的集群都可使用该功能，参加“8.3.2 Resource-level fencing使用集群信息库（CIB）”**

|  |
| --- |
| 重要  绝对至关重要的是至少配置两个独立的集群通信通道此功能正常工作。基于心跳Pacemaker集群应在ha.cf配置文件中定义至少两个集群通信链路。 Corosync集群中至少在corosync.conf有两个冗余环。 |

**8.3.1 Resource-level fencing和dopd**

在基于心跳的Pacemaker集群中，drbd可使用一个叫做outdate-peer 服务的resources-level fencing功能，简称dopd。

**8.3.1.1  dopd的Heartbeat配置**

为启用dopd，必须在文件/etc/ha.d/ha.cf file中添加如下行：

respawn hacluster /usr/lib/heartbeat/dopd

apiauth dopd gid=haclient uid=hacluster

可根据首选的分布调整dopd的路径。在某些分布和结构中，正确的路径为/usr/lib64/heartbeat/dopd。

在进行修改并拷贝ha.cf文件到对等节点后，将Pacemaker置为维护模式，并运行/etc/init.d/heartbeat重新加载配置文件。之后，需要确认dopd为一个正在运行的进程。

|  |
| --- |
| 备注  可使用ps ax | grep dopd 或者是发出 killall -0 dopd.检查这个进行。 |

**8.3.1.2 Dopd的drbd配置**

一旦dopd运行，请为drbd的资源添加如下配置：

resource <resource> {

    handlers {

        fence-peer "/usr/lib/heartbeat/drbd-peer-outdater -t 5";

        ...

    }

    disk {

        fencing resource-only;

        ...

    }

    ...

}

对于dopd而言分布可能将drbd-peer-outdater binary根据系统架构的情况放置到/usr/lib64/heartbeat。

最后，复制drbd.conf到对等节点，并使用drbdadm命令重新调整资源并反映出变化调整。

**8.3.1.3 测试dopd功能**

为了测试dopd是不是运行正常，在心跳服务正常运行时中断配置的复制连接和连接的资源。也可以通过简单的拔出物理链路，当然这是相当有侵略性的。当然也可以通过插入一条临时的iptables规则，丢弃drbd的传输到资源使用的tcp端口的信息。

再次之后，可以观察到资源的连接状态从connection转变为WFConnection。几秒之后，就可以发现磁盘的状态变成了Outdated/DUnknown。而这正是dopd所负责的。

再次之后的任何企图切换过时资源到主角色的操作都会失败。

当重新建立网络连接（不管是插入物理链路还是将iptables的临时规则移除），连接状态都会改变为connection，并自动同步到目标（如果主节点在网络中断期间发生了变化）。然后就观察到一个简短的同步期间，最后，以前过时的资源则再次被标记为UpToDate。

**8.3.2 Resource-level fencing使用集群信息库（CIB）**

需在drbd.conf中设置两个选项才能启用Pacemaker的resource-level fencing：

resource <resource> {

  disk {

    fencing resource-only;

    ...

  }

  handlers {

    fence-peer "/usr/lib/drbd/crm-fence-peer.sh";

    after-resync-target "/usr/lib/drbd/crm-unfence-peer.sh";

    ...

  }

  ...

}

因此，如果drbd复制连接变成disconnected，脚本crm-fence-peer.sh会联系几圈管理器，探测Pacemaker的Master/Slave资源相关的drbd资源，并确保Master/Slave资源不再在任一节点上被提升为当前活跃的。相反，当连接重新建立drbd完成同步后，这个约束就会被删除，并且集群管理器也可自由的在任何节点上升级资源。

**8.4 Pacemaker集群只用堆叠的drbd资源**

Stacked堆叠资源允许drbd被用在多点集群的多级冗余，或建立异地灾备的功能。本节将介绍如何在泽中配置中配置drbd和Pacemaker。

**8.4.1 添加站点灾难恢复和Pacemaker集群**

在这种配置的情况下，会在一个站点两个节点的高可用性集群，外加一个被安置异地处理单独的节点。第三个节点作为一个灾难恢复节点是一个独立的服务器。考虑下面的插图来描述这一概念。

图8.1。在Pacemaker集群中的DRBD资源堆积

在这个例子中，alice和bob为两个Pacemaker 集群节点，而charlie则为一个不被Pacemaker管理的场外节点。

要建立这样的一个配置，首先需要配置和初始化6.15节“创建三节点的设置”中所述的drbd的资源。之后，根据CRM的配置配置Pacemaker：

primitive p\_drbd\_r0 ocf:linbit:drbd \

        params drbd\_resource="r0"

primitive p\_drbd\_r0-U ocf:linbit:drbd \

        params drbd\_resource="r0-U"

primitive p\_ip\_stacked ocf:heartbeat:IPaddr2 \

        params ip="192.168.42.1" nic="eth0"

ms ms\_drbd\_r0 p\_drbd\_r0 \

        meta master-max="1" master-node-max="1" \

        clone-max="2" clone-node-max="1" \

        notify="true" globally-unique="false"

ms ms\_drbd\_r0-U p\_drbd\_r0-U \

        meta master-max="1" clone-max="1" \

        clone-node-max="1" master-node-max="1" \

        notify="true" globally-unique="false"

colocation c\_drbd\_r0-U\_on\_drbd\_r0 \

        inf: ms\_drbd\_r0-U ms\_drbd\_r0:Master

colocation c\_drbd\_r0-U\_on\_ip \

        inf: ms\_drbd\_r0-U p\_ip\_stacked

colocation c\_ip\_on\_r0\_master \

        inf: p\_ip\_stacked ms\_drbd\_r0:Master

order o\_ip\_before\_r0-U \

        inf: p\_ip\_stacked ms\_drbd\_r0-U:start

order o\_drbd\_r0\_before\_r0-U \

        inf: ms\_drbd\_r0:promote ms\_drbd\_r0-U:start

如果创建一个名为/tmp/crm.txt的临时文件，可以使用如下命令导入活动集群的配置：

crm configure < /tmp/crm.txt

该配置将确保在alice/ bob集群的以下动作按照正确的顺序发生：

1、  Pacemaker在两个集群节点上启动drbd资源r0，并升级其中一个节点为主角色。

2、  Pacemaker之后启动ip地址192.168.42.1，用于复制堆叠的资源到第三节点上。先这样做的节点的drbd资源r0晋升为主节点。

3、  在r0的主节点上复制r0-U的ip地址，现在Pacemaker启动r0-U的用于连接和复制到异地节点的drbd资源。

4、  然后Pacemaker提升r0-U的资源为主角色，它可被一个应用程序使用。

因此，Pacemaker的配置确保群集节点之间不仅有完整的数据冗余，而且第三，异地节点也有。

|  |
| --- |
| 备注  这个类型的设置通常和drbd proxy部署在一起。 |

**8.4.2使用堆叠的资源，实现4路冗余的Pacemaker集群**

在此配置中共有三个drbd资源（两个不堆叠的，一个堆叠的），用于实现4路存储冗余。这就意味着一个4节点的集群，即使三个节点同时出现故障，也能保证提供正常的服务。

使用下图来解释这个概念。

图8.2 Pacemaker集群中的DRBD堆叠资源

在这个例子中，alice, bob, charlie,和 daisy组成两个量节点的Pacemaker集群。Alice和bob组成的集群名为左，并复制两点之间的drbd资源，而charlie和daisy在另外一个独立的单元中用途是一样的，名字为right。而堆叠的drbd资源则连接两个集群。

|  |
| --- |
| 备注  由于Pacemaker集群管理器的限制，需要是Pacemaker v1.0.5，这是不可能创建单个4节点集群而不启用CIB功能，这是一个超级进程，不建议一般用途使用。这个可能会在新发布的Pacemaker中解决。 |

为建立这样的配置，首先需要按照6.15节中“创建三节点的设置”配置和初始化资源（除了远程一般的drbd配置为堆叠，而不是本地集群）。之后配置Pacemaker使用下面的CRM配置，并启动left集群：

primitive p\_drbd\_left ocf:linbit:drbd \

        params drbd\_resource="left"

primitive p\_drbd\_stacked ocf:linbit:drbd \

        params drbd\_resource="stacked"

primitive p\_ip\_stacked\_left ocf:heartbeat:IPaddr2 \

        params ip="10.9.9.100" nic="eth0"

ms ms\_drbd\_left p\_drbd\_left \

        meta master-max="1" master-node-max="1" \

        clone-max="2" clone-node-max="1" \

        notify="true"

ms ms\_drbd\_stacked p\_drbd\_stacked \

        meta master-max="1" clone-max="1" \

        clone-node-max="1" master-node-max="1" \

        notify="true" target-role="Master"

colocation c\_ip\_on\_left\_master \

        inf: p\_ip\_stacked\_left ms\_drbd\_left:Master

colocation c\_drbd\_stacked\_on\_ip\_left \

        inf: ms\_drbd\_stacked p\_ip\_stacked\_left

order o\_ip\_before\_stacked\_left \

        inf: p\_ip\_stacked\_left ms\_drbd\_stacked\_left:start

order o\_drbd\_left\_before\_stacked\_left \

        inf: ms\_drbd\_left:promote ms\_drbd\_stacked\_left:start

如果创建一个名为/tmp/crm.txt的临时文件，可以使用如下命令导入活动集群的配置：

crm configure < /tmp/crm.txt

在加入这些配置到CIB后，Pacemaker会执行以下动作：

1、  在alice和bob间启动drbd资源left复制并提升资源角色为主。

2、  启动IP地址10.9.9.100（Alice或Bob，取决于这些持有的left资源的主角色）。

3、  启动drbd资源堆叠在刚刚配置IP地址的同一节点上。

4、  提升堆叠的drbd资源为主角色。

现在继续配置集群right，通过创建以下配置：

primitive p\_drbd\_right ocf:linbit:drbd \

        params drbd\_resource="right"

primitive p\_drbd\_stacked ocf:linbit:drbd \

        params drbd\_resource="stacked"

primitive p\_ip\_stacked\_right ocf:heartbeat:IPaddr2 \

        params ip="10.9.10.101" nic="eth0"

ms ms\_drbd\_right p\_drbd\_right \

        meta master-max="1" master-node-max="1" \

        clone-max="2" clone-node-max="1" \

        notify="true"

ms ms\_drbd\_stacked p\_drbd\_stacked \

        meta master-max="1" clone-max="1" \

        clone-node-max="1" master-node-max="1" \

        notify="true" target-role="Slave"

colocation c\_drbd\_stacked\_on\_ip\_right \

        inf: ms\_drbd\_stacked p\_ip\_stacked\_right

colocation c\_ip\_on\_right\_master \

        inf: p\_ip\_stacked\_right ms\_drbd\_right:Master

order o\_ip\_before\_stacked\_right \

        inf: p\_ip\_stacked\_right ms\_drbd\_stacked\_right:start

order o\_drbd\_right\_before\_stacked\_right \

        inf: ms\_drbd\_right:promote ms\_drbd\_stacked\_right:start

在加入这些配置到CIB后，Pacemaker会执行以下动作：

1、  在charlie 和daisy间启动drbd资源right复制并提升资源角色为主。

2、  启动IP地址10.9.9.100（charlie 或 daisy，取决于这些持有的right资源的主角色）。

3、  启动drbd资源堆叠在刚刚配置IP地址的同一节点上。

4、  保留堆叠stacked drbd资源在次角色（因为target-role="Slave"）

**8.5配置drbd在两个SAN-backed进行复制的Pacemaker集群**

这个一个在split-site中经常采用的一个高级设置。它涉及到两个独立的Pacemaker集群，每个集群访问一个单独的网络存储区域（SAN）。Drbd通过一个两点site间的IP link使用存在SAN上的复制的数据。

使用下图来解释这个概念。

图8.3 基于SAN集群的drbd复制

在每个site的单个节点目前作为drbd对等节点是没有明确定义的--drbd对等节点是浮动的，也就是说，drbd绑定在虚拟ip地址上而不是特定的无理机上。

|  |
| --- |
| 备注  这种类型的设置通常是和drbd proxy或者是基于传输的复制结合在一起的。 |

由于这种类型的设置处理共享存储，为保证它能正常的工作，配置和测试STONITH是绝对至关重要的。

**8.5.1 drbd的资源配置**

需要在drbd.conf中配置以下方式使drbd资源浮动：

resource <resource> {

  ...

  device /dev/drbd0;

  disk /dev/sda1;

  meta-disk internal;

  floating 10.9.9.100:7788;

  floating 10.9.10.101:7788;

}

Floating这个关键字通常置于资源配中的<host>部分。在这种模式下，drbd表示ip地址和tcp端口，而不是对等节点的名字。需要注意的是指定的地址必须为虚拟的集群的IP地址，而不是物理节点的IP地址，对于浮动的节点这点很重要。在这个例子中，在split-site配置两个浮动的地址属于两个独立的IP网络是必须的。要特别注意要允许路由器或防火墙在drbd节点之间进行复制和传输。

**8.5.2 Pacemaker资源的配置**

一个drbd浮动节点的设置，在Pacemaker配置方面，包括如下项目（在两个参与的每一个Pacemaker集群）：

**一个虚拟的集群IP**

**一个主/从的drbd资源（使用drbd OCF资源代理）**

**确保Pacemaker以正确的顺序在正确的节点上上启动**

要在一个浮动的对等节点的2集群节点上配置一个名为mysql的资源，使用复制ip地址10.9.9.100，使用一下crm命令配置Pacemaker：

crm configure

crm(live)configure# primitive p\_ip\_float\_left ocf:heartbeat:IPaddr2 \

                    params ip=10.9.9.100

crm(live)configure# primitive p\_drbd\_mysql ocf:linbit:drbd \

                    params drbd\_resource=mysql

crm(live)configure# ms ms\_drbd\_mysql drbd\_mysql \

                    meta master-max="1" master-node-max="1" \

                         clone-max="1" clone-node-max="1" \

                         notify="true" target-role="Master"

crm(live)configure# order drbd\_after\_left \

                      inf: p\_ip\_float\_left ms\_drbd\_mysql

crm(live)configure# colocation drbd\_on\_left \

                      inf: ms\_drbd\_mysql p\_ip\_float\_left

crm(live)configure# commit

bye

在加入这些配置到CIB后，Pacemaker会执行以下动作：

1、  启用ip地址10.9.9.100（在alice或者是Bob上）

2、  根据所配置的ip地址调用drbd资源。

3、  提升其中一个drbd资源为主角色。

接着，为了创建与集群相配的配置，使用以下命令配置Pacemaker：

crm configure

crm(live)configure# primitive p\_ip\_float\_right ocf:heartbeat:IPaddr2 \

                    params ip=10.9.10.101

crm(live)configure# primitive drbd\_mysql ocf:linbit:drbd \

                    params drbd\_resource=mysql

crm(live)configure# ms ms\_drbd\_mysql drbd\_mysql \

                    meta master-max="1" master-node-max="1" \

                         clone-max="1" clone-node-max="1" \

                         notify="true" target-role="Slave"

crm(live)configure# order drbd\_after\_right \

                      inf: p\_ip\_float\_right ms\_drbd\_mysql

crm(live)configure# colocation drbd\_on\_right

                      inf: ms\_drbd\_mysql p\_ip\_float\_right

crm(live)configure# commit

bye

在加入这些配置到CIB后，Pacemaker会执行以下动作：

1、  启用ip地址10.9.9.101（在charlie或者是daisy上）

2、  根据所配置的ip地址调用drbd资源。

3、  保留drbd资源为次角色（是由target-role="Slave"确定的）

**8.5.3 Site的故障转移**

在split-site配置中服务从一个site转移到另外一个是非常必要的。这可能是一个预定的过渡也可能是一个有灾难性后果的事件。如果是过渡期则是非常正常的，可以预期的时间，我们所期望也当然是这样：

**在连接到集群的site上关于放弃资源，改变影响资源的DRBD target-role，性要从主到从。这将关闭任何依赖的主要角色DRBD资源，降级并继续运行，并准备从新的主上接受更新。**

**连接到集群site上的接替资源，改变受影响的DRBD资源的目标角色的属性从到主。这将升级DRBD资源，启动任何其他Pacemaker的资源，根据DRBD资源的主角色，并复制到远程site的更新。**

**要故障切换回来，就是过程的反过来操作。**

如果在活动的site上发生灾难性停电，site显而易见的会置于offline，而不在复制到备份的site上去。在这种情况下：

**连接到集群上仍在运行的site资源，并改变受影响的DRBD资源的目标角色的属性从到主。这将升级DRBD资源，并根据DRBD资源主角色启动任何其他Pacemaker的资源。**

**当原来的site被恢复或重建，可能会再次连接的DRBD资源，并随后故障恢复使用相反的过程。**

**9 红帽集群集成drbd**

目录

[9.1红帽集群的背景资料](http://write.blog.csdn.net/postedit/6460814#_9.1红帽集群的背景资料)

[9.1.1. Fencing](http://write.blog.csdn.net/postedit/6460814#_9.1.1._Fencing)

[9.1.2资源组管理](http://write.blog.csdn.net/postedit/6460814#_9.1.2资源组管理)

[9.2 红帽集群配置](http://write.blog.csdn.net/postedit/6460814#_9.2_红帽集群配置)

[9.2.1 cluster.conf 文件](http://write.blog.csdn.net/postedit/6460814#_9.2.1_cluster.conf_文件)

[9.3在红帽fail-over集群中使用drbd](http://write.blog.csdn.net/postedit/6460814#_9.3在红帽fail-over集群中使用drbd)

本章介绍了如何在红帽集群高可用性集群中使用DRBD的复制存储。

|  |
| --- |
| 备注  本指南使用非官方的红帽集群，是指其在历史上的多个正式的产品名称，包括红帽集群套件，红帽企业Linux的高可用性附加产品。 |

**9.1红帽集群的背景资料**

**9.1.1. Fencing**

红帽集群，最初设计是为了共享存储集群，凭借节点的fencing放置并发、不协调的连接访问共享资源。红帽集群的fencing功能凭借fencing服务fenced和fencing代理，以脚本的形式实现的。

即使基于DRBD集群没有利用共同的存储资源，从而从drbd角度来看fencing不是必须的，红帽集群套房甚至仍然需要fencing基于drbd的配置。

**9.1.2资源组管理**

资源组管理（rgmanager或clurgmgr）类似于Pacemaker。它是集群管理套件和配置管理的应用程序的主要接口。

**9.1.2.1 红帽集群资源**

在红帽集群术语中单个的高可用应用程序、文件系统、IP地址和类似的资源。

当资源上相互依赖，例如，NFS导出取决于正在安装的文件系统-它们形成一个资源树，这是一种嵌套树资源中的另一个。内嵌套层资源可以承受参数从资源在外层嵌套层。在Pacemaker中不存在资源树的概念。

**9.1.2.2 红帽集群服务**

资源形成一个相互依存的集合，该集合成为服务。这不同于Pacemaker，这样的集合成为一个资源组。

**9.1.2.3 rgmanager资源代理**

资源代理被rgmanager调用，类似于在Pacemaker上的使用，在这个意义上，它们使用基于同一个在开放集群框架（OCF）所界定的shell api，虽然Pacemaker采用了没有被定义的框架的扩展。因此从理论上讲，资源代理和Pacemaker在一定程度是可以互换的—然而，在两个集群管理实际使用不同的但功能和任务相似资源代理。

红帽集群资源代理安装在/usr/share/cluster下。和Pacemaker OCF资源代理的自己包含本身不同，一些红帽集群资源代理被分割到包含a.sh文件的shell代码和a.metadata文件包含xml资源代理元数据的文件。

Drbd包含红帽集群资源代理。它以drbd.sh和drbd.metada的形式安装到传统的目录里面。

**9.2 红帽集群配置**

本节概述获得红帽集群运行所需的必要的配置步骤。准备集群配置是比较简单的，所有基于drbd的红帽集群都需两个节点（即为在红帽文档中所称的集群成员）和fencing设备。

|  |
| --- |
| 备注  欲了解更多有关配置红帽集群的信息，请参阅红帽的红帽集群和GFS的文档。 |

**9.2.1 cluster.conf 文件**

RHEL集群配置保存在一个单独的配置文件/etc/cluster/cluster.conf中，可通过以下方面管理集群配置：

**直接编辑配置文件。**这是最简单的的方法。不需要出文本编辑之外的其他先决条件。

**使用GUI系统配置集群。**这是写在Python中使用Glade的一个GUI应用程序。它需要X显示（无论是直接在服务器控制台，或通过SSH隧道）的可用性。

**使用基于web管理基础设施conga。**Conga的汲取设施包含一个节点代理与本地集群管理器、集群资源管理器以及集群LVM守护进程、页面管理应用程序（luci）。可用一个简单的web浏览器来配置集群的通信。

**9.3在红帽fail-over集群中使用drbd**

|  |
| --- |
| 备注  本节专门对红帽集群fail-over设置drbd，而不涉及GFS。GFS（和GFS2）的配置，参见第11章“使用GFS整合drbd” |

这部分类似于低8章，整合drbd和Pacemaker集群，假定使用如下配置参数配置高可用mysql数据库：

**drbd的资源被用为名为mysql的数据库存储，并管理设备/dev/drbd0.**

**drbd设备拥有一个被挂载/var/lib/mysql下的ext3文件系统（默认为mysql的数据目录）**

**mysql数据库利用文件系统，并监听在集群IP地址192.168.42.1上。**

**9.3.1设置群集配置**

要配置高可用mysql数据库，创建或者是修改包含以下配置项的/etc/cluster/cluster.conf文件。

打开/etc/cluster/cluster.conf文件，然后再资源中配置下列项目：

<rm>

  <resources />

  <service autostart="1" name="mysql">

    <drbd name="drbd-mysql" resource="mysql">

      <fs device="/dev/drbd/by-res/mysql/0"

          mountpoint="/var/lib/mysql"

          fstype="ext3"

          name="mysql"

          options="noatime"/>

    </drbd>

    <ip address="10.9.9.180" monitor\_link="1"/>

    <mysql config\_file="/etc/my.cnf"

           listen\_address="10.9.9.180"

           name="mysqld"/>

  </service>

</rm>

|  |
| --- |
| 备注  例子假设为一个单卷的资源。 |

内嵌套资源引用另外一个<service/>是红帽集群的表达方式资源依赖关系

务必增加config\_version属性，在配置完成之后，在root中找到 <cluster>元素。然后发出以下命令修改正在运行的集群配置：

ccs\_tool update /etc/cluster/cluster.conf

cman\_tool version -r <version>

在第二个命令中，注意需要将<version>替换为你配置的最新的版本号。

|  |
| --- |
| 备注  无论是system-config-cluster GUI配置还是conga基于web集群管理的设施会控告在cluster.conf文件中涉及drbd资源代理的集群配置。这是因为这个两个应用程序并不支持第三方扩展到集群基础设施提供的集群管理的Python管理包。 |

这样，当利用集群配置的drbd资源代理时，它是不建议使用system-config-cluster和Conga集群配置的。使用这些工具只监控集群的状态，以保证它的正常工作。

**10 使用drbd结合LVM**

目录

[10.1 LVM引子](http://write.blog.csdn.net/postedit/6460814#_10.1_LVM引子)

[10.2使用逻辑卷（LV）作为drbd的后端设备](http://write.blog.csdn.net/postedit/6460814#_10.2使用逻辑卷（LV）作为drbd的后端设备)

[10.3 DRBD的同步过程中使用自动化LVM快照](http://write.blog.csdn.net/postedit/6460814#_10.3_DRBD的同步过程中使用自动化LVM快照)

[10.4 使用物理卷PV配置drbd资源](http://write.blog.csdn.net/postedit/6460814#_10.4_使用物理卷PV配置drbd资源)

[10.5 在现有卷组中添加一个新的DRBD卷](http://write.blog.csdn.net/postedit/6460814#_10.5_在现有卷组中添加一个新的DRBD卷)

[10.6 嵌套的LVM配置与DRBD](http://write.blog.csdn.net/postedit/6460814#_10.6_嵌套的LVM配置与DRBD)

[10.7 高可用LVM Pacemaker](http://write.blog.csdn.net/postedit/6460814#_10.7_高可用LVM_Pacemaker)

本章涉及与管理DRBD技术结合的LVM2。特别是涵盖：

**使用LVM逻辑卷作为drbd的后端设备；**

**使用LVM物理卷的drbd设备；**

**结合这些概念，使用drbd实现一个使用LVM分层方法。**

如果不熟悉这些，从10.1节“LVM引子“可作为LVM的起点，同时也鼓励通过自己的方式了解除本节提供的以外的LVM的知识。

**10.1 LVM引子**

LVM2是逻辑卷管理的linux设备映射框架的背景下实施的。与原来的LVM出了名称和首字母缩写意外是没有任何的共同之处的。老的（现在追溯命名为“LVM1”）是过时了的，这在本节中未涉及。

使用LVM时，重要的是要了解其最基本的概念：

**物理卷（PV）。**一个物理卷（PV）是一个完全由LVM管理的底层块设备。PV可是单个磁盘或者是单个分区。常见是在磁盘上创建一个分区专门被linux的LVM使用。

|  |
| --- |
| 备注  这个分区类型“Linux LVM”（签名 0x8e）可用于识别单独使用的LVM分区。然而，这不是必须的—LVM认可物理卷设备初始化后的书面签名的方式。 |

**卷组（VG）。**一个卷组（VG）是LVM的基本管理单元。一个VG可能包括一个或多个几的PV。每个VG都有一个唯一的名称。VG可以在运行时添加额外的PV，或扩大现有的PV。

**逻辑卷（LV）。**LV可在运行期间的VG内创建，并且对内核的其他部分可视为常规块设备。因此，它们可用来保存一个文件系统，或者用于其他任何目的的块设备。LV可以在线调整大小，也可以从一个PV转移到另外一个上（只要PV属于相同的VG）。

**快照逻辑卷（SLV）。**快照是暂时的LV的时间点副本。创建快照几乎是瞬间就完成的，即使原来的LV（原点卷）有大小数百GiByte。通常情况下，快照需要的空间大大低于原来的LV。

**图10.1。 LVM的概述**

**10.2使用逻辑卷（LV）作为drbd的后端设备**

在linux上由于逻辑卷（LV）是一个简单的块设备，因此它可作为drbd设备使用。要以这种方式使用LV，只需要创建然后像其他的设备一样进行初始化即可。

在这个例子中假设卷组（vg）名为foo已经在存在在两个LVM系统的节点上，就可以在该VG上使用LVM创建一个名为r0的drbd资源。

首先，创建逻辑卷：

lvcreate --name bar --size 10G foo

 Logical volume "bar" created

当然，需要在两个drbd集群的节点上都运行以上命令，之后在两个节点上都应该会产生一个名为/dev/foo/bar的块设备。

然后，可以在资源中配置中输入新创建的卷：

resource r0 {

  ...

  on alice {

    device /dev/drbd0;

    disk   /dev/foo/bar;

    ...

  }

  on bob {

    device /dev/drbd0;

    disk   /dev/foo/bar;

    ...

  }

}

现在可以像没有LVM块设备一样继续启动资源。

**10.3 DRBD的同步过程中使用自动化LVM快照**

尽管drbd是同步的，但是SyncTarget同步目标的状态会直到同步完成前都会是Inconsistent状态。如果在这种情况下SyncSource同步源发生故障（无法修复），则将是非常不幸的。有好数据的节点是死的，而有坏数据的节点是存活的。

drbd使用LVM逻辑卷可通过创建一个自动化的快照启动同步，并删除成功同步后的相同的快照，从而减轻这个问题带来的损失。

为了使用快照自动同步，需要在资源中添加如下配置：

**DRBD自动化同步前的快照。**

resource r0 {

  handlers {

    before-resync-target "/usr/lib/drbd/snapshot-resync-target-lvm.sh";

    after-resync-target "/usr/lib/drbd/unsnapshot-resync-target-lvm.sh";

  }

}

两个脚本解析$DRBD\_RESOURCE$环境变量，drbd将之传递给任何它调用的处理程序。snapshot-resync-target-lvm.sh脚本在同步开始前创建一个包含所有资源的LVM的快照，在这种情况下，如果脚本失败，则不进行同步。

一旦同步完成，unsnapshot-resync-target-lvm.sh脚本自动删除不需要快照，如果删除快照失败，快照将继续存在。

|  |
| --- |
| 重要  应该尽快审查摇摆不定的快照。一个完整的快照会导致快照本身和来源的失败。 |

任何时候出现SyncSource同步源不能修复的故障，并且你决定恢复到最新的对等节点的最近的快照，就可使用使用的lvconvert –M命令。

**10.4 使用物理卷PV配置drbd资源**

为了准备一个使用物理卷作为DRBD资源，必须在DRBD设备上创建一个PV签名。为了做到这一点，在主角色的节点上运行一下命令：

pvcreate /dev/drbdX

或者是

pvcreate /dev/drbd/by-res/<resource>/0

|  |
| --- |
| 备注  本例中假设是一个单卷的资源。 |

目前LVM扫描包括该设备在内的PV签名是必需的。为了重要做，需要编辑LVM的配置文件，其通常为/etc/lvm/lvm.conf、找到包括过滤关键字在内的设备部分进行在线编辑，如果所有的PV都存储在drbd设备上，下面则为一个合适的过滤选项：

filter = [ "a|drbd.\*|", "r|.\*|" ]

这种过滤器表示接受任何在DRBD设备上找到的签名，而拒绝其他的。

|  |
| --- |
| 备注  默认情况下，LVM扫描所有在/dev的PV签名的所有块设备。这相当于filter = [ "a|.\*|" ]. |

如果想使用堆叠的资源为LVM的PV，那就需要一个明确的过滤器配置。需要确保LVM在堆叠的资源上探测到PV签名，而忽略与之对应的地产的资源和后端块设备。本例假设底层的drbd资源使用设备从0到9，而在堆叠的资源上则是10以上的：

filter = [ "a|drbd1[0-9]|", "r|.\*|" ]

这种过滤器表示只接受在drbd设备/dev/drbd10到/dev/drbd19以内的PV签名，而忽略其他的。

在修改lvm.conf文件后，必须运行vgscan命令使LVM丢弃他的配置缓存并重新扫描设备的PV签名。

当然也可以使用不同的过滤器配置，以满足特定系统的配置。但是千万要注意：

**接受（含）你希望作为PV的drbd设备**

**拒绝（不包含）相应底层的设备，以避免LVM发现重复的PV签名。**

此外，通过设置禁用LVM缓存：

write\_cache\_state = 0

急用LVM缓存后，确保要删除/etc/lvm/cache/.cache.中任何过时的缓存。

必须对等节点上重复上述步骤。

当配置新的PV后，可以继续将它添加到卷组，或从它创建一个新的卷组。 在主节点上注意操作drbd资源是必须的。

vgcreate <name> /dev/drbdX

|  |
| --- |
| 备注  虽然可在一个卷组中混用DRBD和非DRBD卷组，但是这不推荐并且也不具有任何实用价值。 |

当创建VG后，必须使用lvcreate命令操作逻辑卷（与非基于drbd的卷组）

**10.5 在现有卷组中添加一个新的DRBD卷**

有时可能想在VG中添加一个新的drbd支持的物理卷，每当这样做时，一个新的卷就应该添加到现有的资源配置中。这将保留所有的VG中PV的写保真度。

|  |
| --- |
| 重要  如果LVM卷组是由Pacemaker管理的，可参见10.7节“高可用LVM Pacemaker”。它必须在维护模式改变前对drbd进行配置更改。 |

在本例中，扩展包括额外卷的资源的配置：

resource r0 {

  volume 0 {

    device    /dev/drbd1;

    disk      /dev/sda7;

    meta-disk internal;

  }

  volume 1 {

    device    /dev/drbd2;

    disk      /dev/sda8;

    meta-disk internal;

  }

  on alice {

    address   10.1.1.31:7789;

  }

  on bob {

    address   10.1.1.32:7789;

  }

}

确保您的DRBD配置在每个节点上都是相同，然后执行如下命令：

drbdadm new-minor r0 1

这将保证资源r0为新的第一卷。一旦新的卷被添加到复制流，就需要初始化然后将它添加到卷组中：

pvcreate /dev/drbd/by-res/<resource>/1

lvextend <name> /dev/drbd/by-res/<resource>/1

这将新增PV /dev/drbd/by-res/<resource>/1到<name>VG，维护整个VG的写保真度。

**10.6 嵌套的LVM配置与DRBD**

如果有可能，稍微先进点可即使用LV作为drbd的后端设备，又同时使用drbd设备本身作为PV。为提供一个例子可参见如下配置：

**有两个分别名为/dev/sda1和/dev/sda2准备作为物理卷使用的分区**

**两个PV都可作为本地卷组的一部分**

**意图咋该VG中创建一个名为r0的10Gib的LV**

**这个LV将成为drbd资源的本地也命名为r0的备份设备，对应设备未/dev/drbd0**

**该设备将是对另一个名为replicated的VG的唯一PV**

**这个VG包含两个LV，分别名为foo（4GiB）he bar(6GiB)**

为了使此配置，请按照下列步骤操作：

**在/etc/ LVM/lvm.conf中设置一个适当的过滤器选项：**

indexterm:[LVM]indexterm:[filter expression (LVM)]

filter = ["a|sd.\*|", "a|drbd.\*|", "r|.\*|"]

该过滤器表示接受任何scsi和drbd设备上的PV签名，而拒绝其他所有的。

在修改lvm.conf文件后，必须运行vgscan命令使LVM丢弃缓存的配置并重新扫描设备的PV签名。

**通过设置禁用LVM缓存：**

write\_cache\_state = 0

急用LVM缓存后，确保要删除/etc/lvm/cache/.cache.中任何过时的缓存。

必须对等节点上重复上述步骤。

**现在就可初始化两个scsi的分区为PV：**

pvcreate /dev/sda1

Physical volume "/dev/sda1" successfully created

pvcreate /dev/sdb1

Physical volume "/dev/sdb1" successfully created

**下一步是创建本地底层的VG，包括刚才初始化的两个PV：**

vgcreate local /dev/sda1 /dev/sda2

Volume group "local" successfully created

**现在可以创建LV作为drbd的后端设备：**

lvcreate --name r0 --size 10G local

Logical volume "r0" created

**在这里复制所有的步骤到对等节点上**

**然后编辑/etc/drbd.conf创建一个名为r0的资源：**

resource r0 {

  device /dev/drbd0;

  disk /dev/local/r0;

  meta-disk internal;

  on <host> { address <address>:<port>; }

  on <host> { address <address>:<port>; }

}

在创建好新的资源配置后千万要注意将drbd.conf复制到对等节点。

**在此之后，可参见5.4节“首次启用资源”初始化资源（在两个节点上）。**

**之后在一个节点上晋升资源：**

drbdadm primary r0

**现在，在刚晋升的资源节点上，将drbd设备初始化为一个新的物理卷：**

pvcreate /dev/drbd0

Physical volume "/dev/drbd0" successfully created

**创建名为replicated的VG，在同一节点上初始化PV：**

vgcreate replicated /dev/drbd0

Volume group "replicated" successfully created

**最后，在新创建的VG上创建新的LV：**

lvcreate --name foo --size 4G replicated

Logical volume "foo" created

lvcreate --name bar --size 6G replicated

Logical volume "bar" created

逻辑卷foo和bar将以/dev/replicated/foo 和 /dev/replicated/bar的形式出现在本地节点上：

vgchange -a n replicated

0 logical volume(s) in volume group "replicated" now active

drbdadm secondary r0

然后，对等节点上发出以下命令：

drbdadm primary r0

vgchange -a y replicated

2 logical volume(s) in volume group "replicated" now active

在此之后，块设备/dev/replicated/foo 和 /dev/replicated/bar将在对等节点可用。

**10.7 高可用LVM Pacemaker**

对等节点间传输卷组，并提供相应的逻辑卷的过程实现自动化。Pacemaker LVM资源代理正是以此为目的而设计的。

为了管理存在的支持drbd卷组的管理下，在shell中运行以下命令：

**DRBD支持LVM卷组的Pacemaker配置**

primitive p\_drbd\_r0 ocf:linbit:drbd \

  params drbd\_resource="r0" \

  op monitor interval="15s"

ms ms\_drbd\_r0 p\_drbd\_r0 \

  meta master-max="1" master-node-max="1" \

       clone-max="2" clone-node-max="1" \

       notify="true"

primitive p\_lvm\_r0 ocf:heartbeat:LVM \

  params volgrpname="r0"

colocation c\_lvm\_on\_drbd inf: p\_lvm\_r0 ms\_drbd\_r0:Master

order o\_drbd\_before\_lvm inf: ms\_drbd\_r0:promote p\_lvm\_r0:start

commit

当完成该设置后，Pacemaker将自动使r0卷组在任何drbd资源当前为主角色节点上可用。

**11 使用drbd结合GFS**

目录

[11.1 GFS 引子](http://write.blog.csdn.net/postedit/6460814#_11.1_GFS_引子)

[11.2 创建适用于GFS的drbd资源](http://write.blog.csdn.net/postedit/6460814#_11.2_创建适用于GFS的drbd资源)

[11.3 配置LVM认识的drbd资源](http://write.blog.csdn.net/postedit/6460814#_11.3_配置LVM认识的drbd资源)

[11.4 配置支持GFS的集群](http://write.blog.csdn.net/postedit/6460814#_11.4_配置支持GFS的集群)

[11.5 创建一个GFS文件系统](http://write.blog.csdn.net/postedit/6460814#_11.5_创建一个GFS文件系统)

[11.6 使用GFS文件系统](http://write.blog.csdn.net/postedit/6460814#_11.6_使用GFS文件系统)

本章概述以块设备共享全局文件系统（GFS）建立一个drbd资源的必要步骤。它涵盖了GFS和GFS2。

如果在drbd上使用GFS，必须配置drbd为双主模式。

**11.1 GFS 引子**

红帽全局文件系统（GFS）是红帽执行并发访问共享存储文件系统的。因为任何这样的文件系统，GFS允许多个基点访问相同的存储设备，在读写方面不用担心数据损坏。它是通过分布式锁管理器（DLM）管理集群成员的并发访问的。

GFS从一开始涉及的目的就是像传统的共享存储设备使用。因此drbd是完全可能在双主模式下将GFS作为复制的存储设备的。应用程序可能会受益于减少drbd整读写到本地存储，而不是通常配置运行的SAN设备GFS。此外drbd对每个GFS文件系统增加一个额外的物理考虑，从而保证了冗余。

GFS使用LVM的集群感知的变体，称作集群逻辑卷管理器或CLVM。因此，一些并发存在在使用drbd作为GFS的数据存储和使用drbd作为传统LVM的PV之间。

通常GFS文件是集成在红帽的集群管理的框架内的，即红帽集群。本章介绍在红帽集群中使用DRBD和GFS。

GFS、CLVM和红帽集群在RHEL以及从它所衍生出来的CentOS中是可用的。同样对于Debian GNU/Linux也有相应的安装包。本章加上在红帽企业Linux(RHEL)中运行GFS。

**11.2 创建适用于GFS的drbd资源**

由于GFS是一个集群文件系统，期望从所有的集群节点进行并发的读和写，任何用于存储GFS文件系统的drbd资源都必须配置为双主模式。此外，它建议使用drbd的特性自动处理裂脑的自动恢复。将资源在启动后立即切换到主角色是非常必要的。为了做到这一些，如下在资源中配置如下行：

resource <resource> {

  startup {

    become-primary-on both;

    ...

  }

  net {

    allow-two-primaries yes;

    after-sb-0pri discard-zero-changes;

    after-sb-1pri discard-secondary;

    after-sb-2pri disconnect;

    ...

  }

  ...

}

一旦添加了上面的这些新选项，就需要像往常一样初始化资源。由于资源的allow-two-primaries选项设置的为yes，因此能同时晋升两个节点的资源为主角色。

**11.3 配置LVM认识的drbd资源**

GFS使用CLVM，即VLM的集群变种版本，管理GFS使用的块设备。为了drbd使用CLVM，需要配置LVM。

**使用集群锁。可在/etc/lvm/lvm.conf中配置如下选项：**

locking\_type = 3

**扫描drbd设备，使之从新识别基于drbd的物理卷（PV）。这同样适用于传统的（非集群）的LVM。参见10.4节“使用物理卷PV配置drbd资源”了解详细内容。**

**11.4 配置支持GFS的集群**

在创建新的drbd资源并完成初始化配置，在GFS集群的两个节点必须确保并启动一下系统服务：

**cman（它同时启动ccsd和fencend）**

**clvmd.**

**11.5 创建一个GFS文件系统**

为了在双主的drbd资源上创建一个GFS文件系统，就必要先初始化它作为LVM的逻辑卷（LV）。

与传统的相反，非集群感知LVM配置，由于CLVM的集群感知性必须只在一个节点上完成下列步骤：

pvcreate /dev/drbd/by-res/<resource>/0

Physical volume "/dev/drbd<num>" successfully created

vgcreate <vg-name> /dev/drbd/by-res/<resource>/0

Volume group "<vg-name>" successfully created

lvcreate --size <size> --name <lv-name> <vg-name>

Logical volume "<lv-name>" created

|  |
| --- |
| 备注  这个例子假设一个单卷的资源。 |

CLVM会立即通知对等节点这些变化；对等节点发出的LVS（或lvdisplay）将列出新创建的逻辑卷。

现在可以创建实际的文件系统：

mkfs -t gfs -p lock\_dlm -j 2 /dev/<vg-name>/<lv-name>

或者是一个GFS2文件系统：

mkfs -t gfs2 -p lock\_dlm -j 2 -t <cluster>:<name>

/dev/<vg-name>/<lv-name>

在此命令中-j选项是指为GFS所保持的journals的数量。这必须和GFS集群中节点数量相同的，由于drbd不支持对于两个的节点，因此这里设置的值为2.

-t选项只适用于GFS2文件系统，定义所表的名称。其格式如下<cluster>:<name>,这里<cluster>必须匹配在/etc/cluster/cluster.conf中定义的集群名。因此，只有该集群成员被允许使用文件系统。相反<name>是集群中的独特的任意一个文件系统的名称。

**11.6 使用GFS文件系统**

在创建文件紫铜后，必须将之添加到/etc/fstab：

/dev/<vg-name>/<lv-name> <mountpoint> gfs defaults 0 0

而对于GFS2文件系统，只需要更改定义的文件系统的类型即可：

/dev/<vg-name>/<lv-name> <mountpoint> gfs2 defaults 0 0

千万不要忘记以上操作需要在两个节点上进行的。

在此之后，就可以启动GFS服务挂载新的文件系统（在两个节点上）：

service gfs start

从此时起，只要系统启动时drbd配置自动在RHCS服务器和GFS服务前，就可使用GFS文件系统就像在普通的共享存储上设置一样。

**12 使用OCFS2整合DRBD**

目录

[12.1 OCFS2引子](http://write.blog.csdn.net/postedit/6460814#_12.1_OCFS2引子)

[12.2创建一个适用于OCFS2的DRBD资源](http://write.blog.csdn.net/postedit/6460814#_12.2创建一个适用于OCFS2的DRBD资源)

[12.3 创建OCFS2文件系统](http://write.blog.csdn.net/postedit/6460814#_12.3_创建OCFS2文件系统)

[12.4 Pacemaker OCFS2管理](http://write.blog.csdn.net/postedit/6460814#_12.4_Pacemaker_OCFS2管理)

[12.4.1 给Pacemaker增加一个双主模式的资源](http://write.blog.csdn.net/postedit/6460814#_12.4.1_给Pacemaker增加一个双主模式的资源)

[12.4.2 Pacemaker增加对OCFS2的管理能力](http://write.blog.csdn.net/postedit/6460814#_12.4.2_Pacemaker增加对OCFS2的管理能力)

[12.4.3 给Pacemaker添加OCFS2文件系统](http://write.blog.csdn.net/postedit/6460814#_12.4.3_给Pacemaker添加OCFS2文件系统)

[12.4.4 增加要求的Pacemaker约束管理OCFS2文件系统](http://write.blog.csdn.net/postedit/6460814#_12.4.4_增加要求的Pacemaker约束管理OCFS2文件系统)

[12.5 传统的OCFS2管理（不含Pacemaker）](http://write.blog.csdn.net/postedit/6460814#_12.5_传统的OCFS2管理（不含Pacemaker）)

[12.5.1 配置集群支持OCFS2](http://write.blog.csdn.net/postedit/6460814#_12.5.1_配置集群支持OCFS2)

[12.5.2 使用OCFS2文件系统](http://write.blog.csdn.net/postedit/6460814#_12.5.2_使用OCFS2文件系统)

**12.1 OCFS2引子**

Oracle集群文件系统2版本（OCFS2）是有甲骨文公司开发的并发访问共享存储的文件系统。不同于其前任OCFS是专门设计并只适应Oracle数据库的有效载荷，OCFS2是一个通用的实现了大多POSIX的标准的文件系统。ORACLE真正应用集群（RAC）是OCFS2的比较常见的用例，但是OCFS2同样可以使用在负载均衡的NFS集群。

尽管最初设计是使用传统的共享存储设备，OCFS2也同样是适用于部署双主drbd的。从文件系统读取的应用程序可能会受益于降低读延迟，因为DRBD从本地存储读和写，而不是正常运行的SAN设备OCFS2上。另外，DRBD以增加格外的每个文件系统镜像增加OCFS2的冗余，而不是单一的文件系统镜像（也即共享）。

和其他的共享集群文件系统如GFS，OCFS2允许多个节点在读写模式下访问相同的存储设备，而不用担心数据损坏。同时使用分布式锁管理器（DLM）管理集群节点的并发访问。DLM本身的使用一个虚拟文件系统(ocfs2\_dlmfs)，它独立于实际OCFS2文件系统存在于系统。

OCFS2即可以一种内在的集群通信层也可以挂载和卸载的文件系统进行操作来管理集群成员，或者是推迟Pacemaker集群执行这些任务。

OCFS2是在SUSE Linux企业服务器（它主要是支持共享的集群文件系统），centos、Debian GNU/Linux, 和Ubuntu Server Edition可用。Oracle还同样为红帽企业版linux（RHEL）提供了包。本章假定OCFS2是运行在SUSE linux企业服务器系统上的。

**12.2创建一个适用于OCFS2的DRBD资源**

由于OCFS2是一个共享集群文件系统期望于并发读写在所有的节点上，所有的drbd资源都必须配置为双主模式用于存储OCFS文件系统。此外，建议使用drbd的一些特性，配置裂脑自动修复。而且启动之后立即切换到主角色也是必要的。要做到这些可用按照以下方式配置资源：

resource <resource> {

  startup {

    become-primary-on both;

    ...

  }

  net {

    # allow-two-primaries yes;

    after-sb-0pri discard-zero-changes;

    after-sb-1pri discard-secondary;

    after-sb-2pri disconnect;

    ...

  }

  ...

}

在初始化配置前是不建议设置allow-two-primaries为yes的，而是应该在初始化资源同步完成后这样做。

一旦你添加了这些选项用于刷新资源的配置，就可正常的初始化资源。当设置该资源的allow-two-primaries选项为yes后，就可在两个节点上都晋升资源为主角色。

**12.3 创建OCFS2文件系统**

现在使用OCFS2的mkfs命令创建文件系统：

mkfs -t ocfs2 -N 2 -L ocfs2\_drbd0 /dev/drbd0

mkfs.ocfs2 1.4.0

Filesystem label=ocfs2\_drbd0

Block size=1024 (bits=10)

Cluster size=4096 (bits=12)

Volume size=205586432 (50192 clusters) (200768 blocks)

7 cluster groups (tail covers 4112 clusters, rest cover 7680 clusters)

Journal size=4194304

Initial number of node slots: 2

Creating bitmaps: done

Initializing superblock: done

Writing system files: done

Writing superblock: done

Writing backup superblock: 0 block(s)

Formatting Journals: done

Writing lost+found: done

mkfs.ocfs2 successful

这将在两个节点上的/dev/drbd0上创建一个OCFS2文件系统，并设置文件系统的标签为ocfs2\_drbd0。也可以在mkfs时指定其他选项，请参阅mkfs.ocfs2系统手册了解详细内容。

**12.4 Pacemaker OCFS2管理**

**12.4.1 给Pacemaker增加一个双主模式的资源**

添加一个存在的双主模式的drbd资源为Pacemaker资源资源，使用如下drm配置：

primitive p\_drbd\_ocfs2 ocf:linbit:drbd \

  params drbd\_resource="ocfs2"

ms ms\_drbd\_ocfs2 p\_drbd\_ocfs2 \

  meta master-max=2 clone-max=2 notify=true

|  |
| --- |
| 重要  注意master-max=2变量，对Pacemaker的主从设置启用双主模式。要求drbd的资源配置allow-two-primaries选项需要设置为yes。否则，Pacemaker会在资源校验时报配置错误。 |

**12.4.2 Pacemaker增加对OCFS2的管理能力**

为了管理OCFS2和内核分布锁管理（DLM），Pacemaker使用一下三种不同的资源代理：

**ocf:pacemaker:controld---Pacemaker的DLM接口**

**ocf:ocfs2:o2cb---Pacemaker对OCFS2集群管理的接口**

**ocf:heartbeat:Filesystem---当配置Pacemaker克隆支持集群文件系统的通用文件系统管理的资源代理。**

使用一下crm配置，在所有的Pacemaker OCFS2集群管理的节点上创建一克隆组的资源：

primitive p\_controld ocf:pacemaker:controld

primitive p\_o2cb ocf:ocfs2:o2cb

group g\_ocfs2mgmt p\_controld p\_o2cb

clone cl\_ocfs2mgmt g\_ocfs2mgmt meta interleave=true

一旦完成这个配置，Pacemaker会在集群所有的节点上启动controld和o2cb资源类型的实例。

**12.4.3 给Pacemaker添加OCFS2文件系统**

虽然在clone模式下Pacemaker使用传统的ocf:heartbeat:Filesystem资源代理管理OCFS2文件系统。为使OCFS2文件系统在Pacemaker的管理下，使用一下crm配置：

primitive p\_fs\_ocfs2 ocf:heartbeat:Filesystem \

  params device="/dev/drbd/by-res/ocfs2/0" directory="/srv/ocfs2" \

         fstype="ocfs2" options="rw,noatime"

clone cl\_fs\_ocfs2 p\_fs\_ocfs2

|  |
| --- |
| 注意  这个例子假设一个单卷的资源 |

**12.4.4 增加要求的Pacemaker约束管理OCFS2文件系统**

为了将所有的OCFS2相关的资源和clone的资源配合在一起，对Pacemaker配置添加一下约束：

order o\_ocfs2 ms\_drbd\_ocfs2:promote cl\_ocfs2mgmt:start cl\_fs\_ocfs2:start

colocation c\_ocfs2 cl\_fs\_ocfs2 cl\_ocfs2mgmt ms\_drbd\_ocfs2:Master

**12.5 传统的OCFS2管理（不含Pacemaker）**

|  |
| --- |
| 重要  在本节中介绍的信息适用于传统系统，在Pacemaker中OCFS2的DLM不可用。它被保存在这里仅供参考。新装置应始终使用Pacemaker的方法。 |

**12.5.1 配置集群支持OCFS2**

**12.5.1.1创建配置文件**

OCFS2使用中央配置文件/etc/ocfs2/cluster.conf。

当创建OCFS2集群，一定要添加hosts到集群配置中。集群相互通信默认端口（7777）通常是可选择的。如果你选择了任何其他的端口，一定要注意选择的drbd端口不能和其他的相冲突。

如果感觉直接编辑clust.conf不方便，也可以使用ocfs2console图像化配置工具进行配置。无论选择那种方法，your /etc/ocfs2/cluster.conf文件都会大致如下：

node:

    ip\_port = 7777

    ip\_address = 10.1.1.31

    number = 0

    name = alice

    cluster = ocfs2

node:

    ip\_port = 7777

    ip\_address = 10.1.1.32

    number = 1

    name = bob

    cluster = ocfs2

cluster:

    node\_count = 2

    name = ocfs2

当配置好集群的配置，使用scp将其拷贝到集群的其他的两个节点。

**12.5.1.1 配置O2CB驱动程序**

======的SUSE Linux Enterprise系统

在SLES中，可能会利用O2CB init脚本的配置选项：

/etc/init.d/o2cb configure

Configuring the O2CB driver.

This will configure the on-boot properties of the O2CB driver.

The following questions will determine whether the driver is loaded on

boot.  The current values will be shown in brackets ('[]').  Hitting

<ENTER> without typing an answer will keep that current value.  Ctrl-C

will abort.

Load O2CB driver on boot (y/n) [y]:

Cluster to start on boot (Enter "none" to clear) [ocfs2]:

Specify heartbeat dead threshold (>=7) [31]:

Specify network idle timeout in ms (>=5000) [30000]:

Specify network keepalive delay in ms (>=1000) [2000]:

Specify network reconnect delay in ms (>=2000) [2000]:

Use user-space driven heartbeat? (y/n) [n]:

Writing O2CB configuration: OK

Loading module "configfs": OK

Mounting configfs filesystem at /sys/kernel/config: OK

Loading module "ocfs2\_nodemanager": OK

Loading module "ocfs2\_dlm": OK

Loading module "ocfs2\_dlmfs": OK

Mounting ocfs2\_dlmfs filesystem at /dlm: OK

Starting O2CB cluster ocfs2: OK

======的Debian GNU/ Linux系统Debian上，配置选项/ etc/init.d/o2cb是不可用的。相反，重新配置OCFS2工具包的驱动：

dpkg-reconfigure -p medium -f readline ocfs2-tools

Configuring ocfs2-tools

Would you like to start an OCFS2 cluster (O2CB) at boot time? yes

Name of the cluster to start at boot time: ocfs2

The O2CB heartbeat threshold sets up the maximum time in seconds that a node

awaits for an I/O operation. After it, the node "fences" itself, and you will

probably see a crash.

It is calculated as the result of: (threshold - 1) x 2.

Its default value is 31 (60 seconds).

Raise it if you have slow disks and/or crashes with kernel messages like:

o2hb\_write\_timeout: 164 ERROR: heartbeat write timeout to device XXXX after NNNN

milliseconds

O2CB Heartbeat threshold: `31`

                Loading filesystem "configfs": OK

Mounting configfs filesystem at /sys/kernel/config: OK

Loading stack plugin "o2cb": OK

Loading filesystem "ocfs2\_dlmfs": OK

Mounting ocfs2\_dlmfs filesystem at /dlm: OK

Setting cluster stack "o2cb": OK

Starting O2CB cluster ocfs2: OK

**12.5.2 使用OCFS2文件系统**

当完成集群配置并创建文件系统，就可像其他的文件系统一样挂载：

mount -t ocfs2 /dev/drbd0 /shared

内核日志（使用dmesg命令）应包含类似的如下行：

ocfs2: Mounting device (147,0) on (node 0, slot 0) with ordered data mode.

在此之前，在读写模式下，应该同时在两个节点上挂载OCFS2文件系统。

**13 使用Xen整合drbd**

目录

[13.1 Xen 引子](http://write.blog.csdn.net/postedit/6460814#_13.1_Xen_引子)

[13.2 Xen下设置drbd的模块参数](http://write.blog.csdn.net/postedit/6460814#_13.2_Xen下设置drbd的模块参数)

[13.3 创建适用于Xen VBD的DRBD资源](http://write.blog.csdn.net/postedit/6460814#_13.3_创建适用于Xen_VBD的DRBD资源)

[13.4 使用DRBD的VBD](http://write.blog.csdn.net/postedit/6460814#_13.4_使用DRBD的VBD)

[13.5 启动，停止和迁移DRBD支持的domU](http://write.blog.csdn.net/postedit/6460814#_13.5_启动，停止和迁移DRBD支持的domU)

[13.6 DRBD/Xen的内部整合](http://write.blog.csdn.net/postedit/6460814#_13.6_DRBD/Xen的内部整合)

[13.7 集成Xen和Pacemaker](http://write.blog.csdn.net/postedit/6460814#_13.7_集成Xen和Pacemaker)

本章概括了在XEN虚拟环境中使用drbd作为虚拟块设备。

**13.1 Xen 引子**

Xen是最初是由英国剑桥大学开发，后由XenSource公司（现在是Citrix的一部分）维护的一个虚拟化的框架。它包含在大多数的linux发行版，如Debian GUN/Linux（从第四版后）、SUSE企业linux服务器（从第十版后）、红帽企业linux（从第五版之后）等等。

Xen使用准虚拟化技术---虚拟化的方法涉及虚拟主机和客户虚拟机之间的高度合作---比传统的虚拟化解决方案（如典型的基于硬件的仿真）提高了选定的客户操作系统的性能。Xen同时还支持适当虚拟化扩展CPU，支持完整的硬件仿真，用Xen的说法，这就是众所周知的HVM（硬件辅助虚拟机）。

|  |
| --- |
| 注意  在写操作时，Xen支持HVM的CPU扩展是Intel的虚拟化技术（VT，以前称为Vanderpool）和AMD的 安全虚拟机（SVM，以前称为Pacifica）。 |

Xen支持实时迁移，这是指从一个屋里主机迁移到来宾操作系统中，而不造成中断。

当drbd资源作为Xen的复制虚拟块设备时，它可用在两台服务器上，然后配置为自动故障转移到一个包含全部内容的domU虚拟磁盘。在这种方式下，DRBD不仅仅在linux服务器下（在非虚拟化的drbd技术的部署方案），而且在其他的任何操作系统都可使用Xen虚拟化技术实现冗余。在本质上，它包括32位和64位intel兼容的框架。

**13.2 Xen下设置drbd的模块参数**

对于Xen Domain-0内核，建议加载drbd模块设置为1.创建或者是打开/etc/modprobe.d/drbd.conf文件，然后输入如下行：

options drbd disable\_sendpage=1

**13.3 创建适用于Xen VBD的DRBD资源**

将drbd资源配置为Xen虚拟化块设备是比较简单的。事实上，典型的drbd配置可用于任何目的。但是，如果想使guest实例实时迁移的话，就需要启用资源的双主模式。

resource <resource> {

  net {

    allow-two-primaries yes;

    ...

  }

  ...

}

对Xen来说是必须启用双主模式的，在启动迁移前，检查所有VBD资源配置的来源和目的地主机的写权限。

**13.4 使用DRBD的VBD**

为使用虚拟块设备做drbd资源，必须在en domU的配置中添加如下行：

disk = [ 'drbd:<resource>,xvda,w' ]

这个配置例子中，命名为resource的drbd资源/dev/xvda在读写模式下对DomU可用。

当然也可是一个单一的domU私用多个drbd资源。这种情况下，需要添加刚那个多的条目，并以分号将磁盘选项隔开。

|  |
| --- |
| 注意  有以下三种情况不能使用这种方法： |

**正在配置一个全虚拟化（HVM）的DomU**

**正使用图形化安装工具安装DomU并且高图形化安装不支持drbd的语法**

**正配置一个DomU，无kernel、initrd和额外的选项，以引导和bootloader\_args使用不支持drbd的语法Xen pseudo-bootloader,pseudo-bootloader。**

pygrub+（之前的Xen3.3）和domUloader.py（使用Xen的SUSE Linux Enterprise Server 10的运）两个伪引导程序不支持DRBD的虚拟块设备的配置语法的例子。

pygrub从Xen3.3开始，domUloader.py版本SLES 11的附带支持这个语法。

在这种情况下，必须使用传统的PHY设备语法和DRBD设备的名称和资源，而不是资源的名称相关联的。然而，这需要管理DRBD之外的Xen，这是一个DRBD资源类型提供的状态转换灵活的方式。

**13.5 启动，停止和迁移DRBD支持的domU**

**启动domU**。当配置DRBD支持domU，就可以像其他的任何DomU一样的方式启动：

xm create <domU>

Using config file "+/etc/xen/<domU>+".

Started domain <domU>

在这个过程中配置为VBD的drbd资源将被晋升为主角色，并能被Xen访问。

**停止domU**。操作简单如下：

xm shutdown -w <domU>

Domain <domU> terminated.

在domU成功关闭后，Drbd资源会转变为次角色：

**迁移domU**。通常使用Xen工具：

xm migrate --live <domU> <destination-host>

这种情况下，自动快速并连续的采取若干管理步骤。晋升目标主机的资源为主角色—本地主机上的实时迁移domU启动---当完成到目标主机的迁移本地主机的资源变成次角色。

事实上资源必须在两个主机上都处于主角色，这也是将资源配置为双主模式的首要原因。

**13.6 DRBD/Xen的内部整合**

Xen本身支持两种虚拟块设备类型：

**Phy。此设备类型用于交互物理块设备，用于主机环境中，以实际透明的方式关闭guest domU。**

**File。次设备类型用于提供给guest domU基于文件的块设备镜像。它的工作原理是创建一个循环块设备，从原始图像文件取出，然后交给该块设备，并以和PHY大致相同的方式关闭domU。**

如果在domU的disk选项中配置一个虚拟块设备使用任何前缀，而不是phy:, file:,或者是没有前缀（这种情况下Xen默认使用PHY设备类型），在名为Xen的scripts目录中可找到比较常用的脚本/etc/xen/scripts。

DRBD分布给drbd设备类型提供一个名为/etc/xen/scripts/block-drbd的脚本。这个脚本处理本章之前所描述的那些必要的drbd的资源状态转换。

**13.7 集成Xen和Pacemaker**

为了充分利用基于DRBD的Xen VBD，推荐心跳管理相关的domU心跳的资源。

可以配置Xen domU为Pacemaker资源，并自动化资源的故障转移。需要使用Xen OCF资源代理实现它。如果已经使用本章所述的DRBD Xen相关的设备类型，将不需要配置单独的drbd资源给Xen集群资源使用。相反，block-drbd辅助脚本会帮你完成所有的必要的资源转换。

当连接时裂脑受害者将立即改变连接状态为SyncTarget，并被主节点覆盖其余节点的修改。

**第五篇 优化DRBD性能**

**14 测试块设备的性能**

目录

[14.1 测量吞吐量](http://write.blog.csdn.net/postedit/6460814#_14.1_测量吞吐量)

[14.2 测量延迟](http://write.blog.csdn.net/postedit/6460814#_14.2_测量延迟)

**14.1 测量吞吐量**

测试DRBD对系统的I/O吞吐量时，对而系统而言绝对吞吐量是无关紧要的。更意思的是drbd对I/O性能的影响是相对的。

|  |
| --- |
| 注意  在本节中所描述的测试是侵入式的，通过覆盖数据导致drbd之间不同步。很重要的是测试要在可以在临时的测试完成后可以丢失的卷上进行。 |

I/O吞吐量的测试是通过写入大块数据到一个设备上，测试系统完成写操作所消耗的时间。可以使用比较实用的工具dd，它在新版本中有相应的内置吞吐量的估计。

下面是一个基于dd吞吐量的简单标准，假设对一个当前连接并在两个节点都处于次角色的名为test临时资源：

TEST\_RESOURCE=test

TEST\_DEVICE=$(drbdadm sh-dev $TEST\_RESOURCE)

TEST\_LL\_DEVICE=$(drbdadm sh-ll-dev $TEST\_RESOURCE)

drbdadm primary $TEST\_RESOURCE

for i in $(seq 5); do

  dd if=/dev/zero of=$TEST\_DEVICE bs=512M count=1 oflag=direct

done

drbdadm down $TEST\_RESOURCE

for i in $(seq 5); do

  dd if=/dev/zero of=$TEST\_LL\_DEVICE bs=512M count=1 oflag=direct

done

本测试只是写一个大小为512M的块到drbd设备，然后进行对比和测试。每个测试都进行5次，然后取平均值。相关结果都是通过dd进行吞吐量的测试的。

|  |
| --- |
| 备注  对新启用的drbd设备，在第一次运行dd时刻明显发现器性能有所下降，这主要是因为并不引人注意的“cold”日志引起的。 |

**14.2 测量延迟**

延迟量和吞吐量是基于完全不同的标准的：在I/O延迟测试中，写一个非常小的数据块（理想情况下，为该系统能处理的最小数据块），并观察完成写入所需的时间。该过程通常反复数次使之在正常的范围波动。

正如吞吐量测试，I/O延迟测试也可使用dd进行测试，尽管有不同的设置以及完全不同的观察重点。

下面是一个基于dd吞吐量的简单标准，假设对一个当前连接并在两个节点都处于次角色的名为test临时资源：

TEST\_RESOURCE=test

TEST\_DEVICE=$(drbdadm sh-dev $TEST\_RESOURCE)

TEST\_LL\_DEVICE=$(drbdadm sh-ll-dev $TEST\_RESOURCE)

drbdadm primary $TEST\_RESOURCE

dd if=/dev/zero of=$TEST\_DEVICE bs=512 count=1000 oflag=direct

drbdadm down $TEST\_RESOURCE

dd if=/dev/zero of=$TEST\_LL\_DEVICE bs=512 count=1000 oflag=direct

在drbd设备上测试写1000个512字节的数据块，之后对后端的设备进行对比。512字节的块大小是linux所支持的做小的块（S390以外的所有架构）。

重要的是要了解到吞吐量产生的测试和dd测试是完全不相干的。重要的进行1000次写所耗费的时间。这段时间除以1000即是单一文件的延迟写时间。

**15 优化drbd的吞吐量**

目录

[15.1 硬件注意事项](http://write.blog.csdn.net/postedit/6460814#_15.1_硬件注意事项)

[15.2 期望的吞吐量开销](http://write.blog.csdn.net/postedit/6460814#_15.2_期望的吞吐量开销)

[15.3 调整建议](http://write.blog.csdn.net/postedit/6460814#_15.3_调整建议)

[15.3.1 设置 max-buffers 和 max-epoch-size](http://write.blog.csdn.net/postedit/6460814#_15.3.1_设置_max-buffers)

[15.3.2 调整I/O unplug watermark](http://write.blog.csdn.net/postedit/6460814#_15.3.2_调整I/O_unplug)

[15.3.3 调整TCP发送缓冲区的大小](http://write.blog.csdn.net/postedit/6460814#_15.3.3_调整TCP发送缓冲区的大小)

[15.3.4 调整活动日志的大小](http://write.blog.csdn.net/postedit/6460814#_15.3.4_调整活动日志的大小)

[15.3.5 禁用barriers和磁盘刷新](http://write.blog.csdn.net/postedit/6460814#_15.3.5_禁用barriers和磁盘刷新)

本章涉及优化DRBD的吞吐量。为实现该目的，本文结合硬件的注意事项提出调整优化和细节吞吐量的建议。

**15.1 硬件注意事项**

Drbd的吞吐量收带宽和底层I/O子系统（磁盘、控制器和相应的缓存），以及复制网络的带宽影响。

**I/O子系统的吞吐量。**I/O子系统的吞吐量是在一定程度上确定的，为以并行写入磁盘的数量。通常单一的、接近合理的、scsi或sas磁盘以40MB/s的流写入单个磁盘。当部署一个条带化配置时，I/O子系统以并行方式写磁盘，有效的多条带化配置则为单个磁盘吞吐量的乘积。因此，40MB/s的磁盘支撑在raid0或者是raid-1+0三条带中120MB/s，或者是5条带的200MB/s。

|  |
| --- |
| 备注  对新启用的drbd设备，在第一次运行dd时刻明显发现器性能有所下降，这主要是因为并不引人注意的“cold”日志引起的。 |

磁盘镜像（RAID-1）对吞吐量的影响对应比较小。校验（RAID- 5）的条带化的磁盘对吞吐量的影响，则往往是不利的。

**网络吞吐量。**网络吞吐量通常是目前网络中以路由/交换机等设置所支持的流量。尽管drbd允许通过专用链接传输，但是这还是主要的drbd复制环节。因此，网络吞吐量改善可通过切换到更好的网络协议（如10G太网）或者通过聚合多个网络的连接绑定到指定的linux的网络驱动设备上。

**15.2 期望的吞吐量开销**

当估算吞吐量和drbd相关的开销时，重要是考虑一下自然的限制：

**drbd的吞吐量受原始I/O子系统的限制。**

**drbd的吞吐量受可用网络带宽的限制。**

两个之间的最小值就是drbd可支持的最大吞吐量。Drbd减少额外的吞吐量开销大约小于最大吞吐量的3%。

**例如，考虑两个集群节点包含I/O子系统能力为200MB/s，之间可以通过一个千兆以太网的链路。千兆以太网可以预计将产生110MB/s的tcp连接的网络吞吐量，从而在这个配置中的瓶颈就是所预计drbd的最大吞吐量为107MB/s。**

**相反，如果I/O子系统的写入能力只有100MB/s，那么它就成为了瓶颈，因此可以预计drbd的最大吞吐量为97MB/s。**

**15.3 调整建议**

Drbd提供的大量的配置选项，这些都可能对系统的吞吐量产生影响。本节中对吞吐量的调整提出一些建议。然后，由于吞吐量在很大程度上依赖于硬件，在这里所描述影响可能对不同的系统有极大的不同。需要了解到这里的建议不是“灵丹妙药”，不能解决所有的吞吐量的瓶颈。

**15.3.1 设置 max-buffers 和 max-epoch-size**

这些选项会影响次节点上写的性能。max-buffers是drbd数据写入到磁盘分配的缓冲区的最大数，而max-epoch-size 的大小是两个写瓶颈的最大允许写的数量。在大多数情况下，应该同时设置两个选项，并设置为相同的值。二者默认为2048，将其设置在8000应该是最合理的利用高性能硬件raid控制。

resource <resource> {

  net {

    max-buffers 8000;

    max-epoch-size 8000;

    ...

  }

  ...

}

**15.3.2 调整I/O unplug watermark**

在正常操作期间I/O unplug watermark是可调整的，它影响I/O子系统控制器多久是“kicked”（被迫进程等待I/O请求）。对该选项的设置没有统一的建议，它的设置很大程度上是依赖于硬件的。

一些控制器在“kecked”频繁时表现比较好，所以这些控制器设置的较低是比较有意义的，甚至于低于drbd允许的最低值（16）。当单独离开时其他表现的比较好，对于这些控制器设置max-buffers高点是可取的。

resource <resource> {

  net {

    unplug-watermark 16;

    ...

  }

  ...

}

**15.3.3 调整TCP发送缓冲区的大小**

TCP发送缓冲区是内存缓冲区传出的TCP的流量。在默认情况下它设置大小为128KiB。适用于高吞吐量网络使用（如专用的千兆以太网或者是负载均衡的连接），它增大到512KiB或者更多是比较有意义的。不推荐发送缓冲区超过2MiB的大小（也不可能带来任何吞吐量的提高）。

resource <resource> {

  net {

    sndbuf-size 512k;

    ...

  }

  ...

}

DRBD还支持发送缓冲区的自动调节。启动该功能后，drbd会动态的选择一个合适的TCP发送缓冲区的大小。TCP发送缓冲区自动调节可以将将缓冲区的大小设置为0.

resource <resource> {

  net {

    sndbuf-size 0;

    ...

  }

  ...

}

**15.3.4 调整活动日志的大小**

如果使用drbd的应用程序写操作比较频繁的写到一个设备上，它通常最好使用一个相当大的活动日志。否则，频繁的元数据更新会影响写性能。

resource <resource> {

  disk {

    al-extents 3389;

    ...

  }

  ...

}

**15.3.5 禁用barriers和磁盘刷新**

|  |
| --- |
| 警告  本节中所述的建议只适用于非易失性（battery backed）控制器缓冲系统。 |

系统配置battery backed write cache是都带内置的手段保护数据以面对停电。这种情况下，处于同样的目的它允许禁用drbd自己的保护措施。这有利于吞吐量的提升。

resource <resource> {

  disk {

    disk-barrier no;

    disk-flushes no;

    ...

  }

  ...

}

**16 优化DRBD的延迟**

目录

[16.1 硬件注意事项](http://write.blog.csdn.net/postedit/6460814#_16.1_硬件注意事项)

[16.2 期望的延迟开销](http://write.blog.csdn.net/postedit/6460814#_16.2_期望的延迟开销)

[16.3 调整建议](http://write.blog.csdn.net/postedit/6460814#_16.3_调整建议)

[16.3.1 设置DRBD的CPU掩码](http://write.blog.csdn.net/postedit/6460814#_16.3.1_设置DRBD的CPU掩码)

[16.3.2 修改网络的MTU](http://write.blog.csdn.net/postedit/6460814#_16.3.2_修改网络的MTU)

[16.3.3 启用I / O期限调度](http://write.blog.csdn.net/postedit/6460814#_16.3.3_启用I_/)

本章涉及优化DRBD的延迟。为实现该目的，本文结合硬件的注意事项提出调整优化和细节延迟的建议。

**16.1 硬件注意事项**

DRBD的延迟收I/O子系统延迟（磁盘、控制器和对应的缓存）和复制网络延迟的影响。

**I/O子系统延迟。**I/O子系统的延迟主要体现在磁盘的转速上，因此利用高转速的磁盘可有效减少I/O子系统的延迟。

同样，使用支持高速写缓存池（BBWC）减少写完成的时间，也能降低写入延迟。最合理的存储子系统与某种形式的高度缓存池，并允许管理员配置词缓存的那一部分用于读取和写入的操作。推荐的方法是禁用磁盘的完全读缓存并对磁盘写缓存都使用内存缓存。

**网络延迟。**网络延迟本质上是数据包在主机之间往返的时间。它是由多项因素影响，其中大多数是专用的、使用与drbd复制连接无关的紧密的网络连接。因此，在千兆以太网间的100到200微秒（μs）RTT秩序报的延迟是可以接受的。

网络延迟可能只能使用低延迟的网络协议降低延迟，如通过Dolphin的SuperSockets在Dolphin Express运行drbd。

**16.2 期望的延迟开销**

当估算结合drbd的延迟的开销时，对吞吐量有一些重要的自然的限制需要考虑：

**drbd延迟受原始I/O子系统的限制。**

**drbd延迟受可用网络带宽的延迟的限制。**

两者之后为drbd的理论的最低延迟。Drbd预计会带来不到1%的额外延迟开销。

**加入本地磁盘子系统写延迟为3ms而网络延迟为0.2ms，那么drbd的延迟以及就为3.2ms或者比仅仅写本地磁盘有大约一个7%的延迟。**

|  |
| --- |
| 备注  延迟还会受到一些其他的因素的影响，包括CPU高速缓存为命中、上下文切换等。 |

**16.3 调整建议**

**16.3.1 设置DRBD的CPU掩码**

DRBD允许其内核线程设置一个明确的CPU掩码，这对程序是特别有利的，否则会与drbd的CPU周期进行竞争。

CPU掩码是其在二进制的的最低有效位表示第一个CPU、第二低表示第二个等等。一个位掩码意味着drbd对应的可用的CPU，否则该位置就必须别清空。例如1（00000001）表示drbd只能使用第一个CPU，而12（00001100）则表示drbd可使用第三和第第四个CPU。

例如，CPU资源的掩码配置如下：

resource <resource> {

  options {

    cpu-mask 2;

    ...

  }

  ...

}

|  |
| --- |
| 重要  当然为了减少drbd与其他程序的CPU竞争，必须配置其他应用程序不使用drbd使用的CPU。 |

某些应用程序也可能提供像drbd这样配置的配置的文件。而其他的应用程序的init脚本中

包括taskset命令。

**16.3.2 修改网络的MTU**

当基于块的（而不是基于Extent的）文件系统出在drbd的上层，可能改变复制网络的最大传输单元（MTU）大小是有益的，其默认为1500字节。通俗的将这就是“启用Jumbo 帧”。

|  |
| --- |
| 备注  基于块的文件系统，包括ext3、ReiserFS（版本3）以及GFS。基于Extent的文件系统，如XFS、 Lustre 和 OCFS2。基于Extent的文件系统只有当拥有大量的文件时启用jubmbo帧就会比较有益。 |

使用下面的命令可以改变的MTU：

ifconfig <interface> mtu <size>

或者

ip link set <interface> mtu <size>

<interface>是指用于DRBD的复制到网络接口。<SIZE>典型值为9000（字节）。

**16.3.3 启用I / O期限调度**

当结合高性能使用时，回写启用硬件的raid控制器，drbd的延迟可能会因使用I/O调度的期限而大大减少，而不是使用CFQ的调度。后者通常在最新的内核配置中（2.6.18之后）默认是启用的。

通过sysfs虚拟文件系统修改I/O调度配置，挂载在/sys下。调度程序配置在/sys/block/<device>中，这里<device>是使用的后端设备。

可通过如下命令启用最后期限调度：

`echo deadline > /sys/block/<device>/queue/scheduler`

然后还可以通过设置下面的提供额外的延迟效益的值：

**禁用前合并：**

echo 0 > /sys/block/<device>/queue/iosched/front\_merges

**减少读取I / O期限为150毫秒（默认为500ms）：**

echo 150 > /sys/block/<device>/queue/iosched/read\_expire

**减少写入I / O的截止日期为1500毫秒（默认为3000ms）：**

echo 1500 > /sys/block/<device>/queue/iosched/write\_expire

如果这些对延迟有所改进，那么要想永久使用的话，也可在系统启动时自动设置。Debian和Ubuntu系统通过sysfsutils和配置文件/etc/sysfs.conf提供该功能。

也可通过内部命令设置一个全局性的I/O调度。编辑引导装载程序配置（如果使用grub引导的haunted，通常在/boot/grub/menu.lst中），同时添加elevator=deadline选项设置内核启动选项。

**第六篇 更多有关drbd的知识**

**17 DRBD的内部**

目录

[17.1 Drbd元数据](http://write.blog.csdn.net/postedit/6460814#_17.1_Drbd元数据)

[17.1.1 内部元数据](http://write.blog.csdn.net/postedit/6460814#_17.1.1_内部元数据)

[17.1.2 外部元数据](http://write.blog.csdn.net/postedit/6460814#_17.1.2_外部元数据)

[17.1.3 元数据大小估算](http://write.blog.csdn.net/postedit/6460814#_17.1.3_元数据大小估算)

[17.2 代标识](http://write.blog.csdn.net/postedit/6460814#_17.2_代标识)

[17.2.1 数据的代](http://write.blog.csdn.net/postedit/6460814#_17.2.1_数据的代)

[17.2.2 代标识的元组](http://write.blog.csdn.net/postedit/6460814#_17.2.2_代标识的元组)

[17.2.3 生成标识符如何改变](http://write.blog.csdn.net/postedit/6460814#_17.2.3_生成标识符如何改变)

[17.2.4 如何DRBD使用代标识符](http://write.blog.csdn.net/postedit/6460814#_17.2.4_如何DRBD使用代标识符)

[17.3 活动日志](http://write.blog.csdn.net/postedit/6460814#_17.3_活动日志)

[17.3.1 目的](http://write.blog.csdn.net/postedit/6460814#_17.3.1_目的)

[17.3.2 活跃程度](http://write.blog.csdn.net/postedit/6460814#_17.3.2_活跃程度)

[17.3.3 选择合适的活动日志的大小](http://write.blog.csdn.net/postedit/6460814#_17.3.3_选择合适的活动日志的大小)

[17.4 快速同步位图](http://write.blog.csdn.net/postedit/6460814#_17.4_快速同步位图)

[17.5 对等节点的fencing接口](http://write.blog.csdn.net/postedit/6460814#_17.5_对等节点的fencing接口)

本章提供一些关于drbd内部算法和结构的背景资料。目的是给那些对背景知识感兴趣的用户提供drbd的背景资料。对drbd的开发不做深入drbd的内部运作做深入的参考。因此请参阅18.6节“出版物”中所累出的文件，其对drbd的源代码进行了注释。

**17.1 Drbd元数据**

一个专门存储drbd各种相关信息的区域。此元数据包括：

**drbd设备的大小**

**代标识（GI，详细信息参见第17.2节“代标识”）**

**活动日志（AL,详细介绍参见第17.3节“活动日志”）**

**快速同步位图（详细参见17.4节“快速同步位图”）**

此元数据可以存储在内部和外部，使用暗中方法是在每个资源的基础上进行配置的。

**17.1.1 内部元数据**

配置资源使用内部元数据意味着drbd存在和实际生成数据相同的底层物理设备上。预留一个区域明确用户存储元数据。

**优点。**由于元数据和实际的数据有着千丝万缕的联系，如果硬盘出现故障管理员没有特殊的操作需求。这样元数据和实际都将一起丢失或者是一起恢复。

**缺点。**如果底层设备是一单个的物理硬盘（没有raid设置），内部元数据可能会对写入吞吐量产生负面影响。应用程序的的写请求可能会触发drbd元数据的更新，如在统一磁盘存储元数据，写操作可能会导致磁盘的读/写有额外开销。

|  |
| --- |
| 重要  如果计划使用内部元数据结合现有的已有数据的底层的块设备，就必须考虑drbd的元数据所需的空间。 |

否则，drbd的资源创建后，新创的元数据将覆盖底层设备的其他元数据，并破坏进程中存在的问题。为了避免这种情况，如下做下面的事情：

**扩展底层设备。**任何逻辑卷管理设施（如LVM）只要有可用的空闲空间提供相应卷的grouplt，就可能被硬件存储解决方案所支持。

**在底层设备上收缩现有的文件系统。**这可能会也可能不会被文件系统所支持。

**如果两者都不能的话，就使用外部元数据替代。**

估算扩展底层设备以及收缩文件系统的大小，可参阅17.1.3节“元数据大小估算”。

**17.1.2 外部元数据**

外部元数据是单独存储、专用的块设备存储的生产数据。

**优点。**使用外部元数据对于写操作的延迟将有所改善。

**缺点。**元数据与实际的生产数据联系不紧密。这意味着，在硬件发生故障，破坏了生产数据（而不是drbd的元数据），需认为干预从幸存节点到被替换节点的数据同步。

如果出现以下情况，外部元数据是唯一可选择的：

**正在使用的drbd复制的设备已经包含了有用的数据。**

**现有设备不支持扩展。**

**设备上的现有文件系统不支持收缩。**

估算扩展底层设备以及收缩文件系统的大小，可参阅17.1.3节“元数据大小估算”。

**17.1.3 元数据大小估算**

可使用下面的公式精确估算出drbd元数据对空间的需求：

图17.1。计算DRBD技术元数据的大小（完全）

Cs是扇区中的数据设备的大小。

|  |
| --- |
| 备注  可以检索设备的大小使用blockdev --getsz <device>命令 |

结果Ms也用扇区表示。要处以2048（一个512字节的扇区大小，这是在S390外所有的linux平台上默认的）转换才MB。

在实践中，可使用下面给出的一个很近似、合理的方式计算。注意在公式中，单位是兆字节，而不是扇区。

**图17.2。估计DRBD技术元数据尺寸（大约）**

**17.2 代标识**

Drbd使用代表示（GI）确定复制数据的“代”。

这是drbd的内部机制，用于：

**确定两个节点是不是属于同一集群的事实（而不是意外连接的两个节点）**

**确定重新同步时同步的方向（如果必要的话）**

**确定是完全重新同步或者是部分重新同步是否满足需求**

**确定裂脑**

**17.2.1 数据的代**

发送下列情况标志着drbd产生一个新的数据代：

**最初设备的完全同步**

**一个断开连接的资源切换到住角色**

**主角色的资源断开连接**

由此可以总结出，只要资源处于连接状态，两个节点的磁盘状态为UpToDate，目前两个节点上数据生成是相同的。反过来也是如此。

每个新的数据生成的标识都由一个8字节的、全局唯一的标识符（UUID）标志。

**17.2.2 代标识的元组**

Drbd保持四个当前的和历史数据代在当地资源元数据的信息：

**当前的UUID。**从本地节点的角度看，这是当前数据代的代标识符。当一个资源连接并完全同步时，目前两节点之间的UUID是相同的。

**位图的UUID。**这是磁盘的同步位图是跟踪更改的代的UUID。只有在断开模式磁盘上的同步位图本身才与该标识符有关。如果资源是连通的，此UUID总是空（零）。

**两个历史UUID。**这是前几代的两个数据的标识符。

总的来说，这四个项目被称为一代标识元组，或简称GI tuple。

**17.2.3 生成标识符如何改变**

**17.2.3.1 开始一个新的数据代**

当一个节点失去与对等节点的连接（无论是网络故障还是人为干预），drbd通过以下方式修改本地的代标识：

**图17.3 GI元组的变化在一个新的数据代开始**

1.     给新的数据代创建一个新的UUID。这将成为当前主节点新的UUID。

2.     之前的UUID现在指代的位图跟踪到变化，它在主节点变成新的位图UUID。

3.     在辅助节点上，GI元组保持不变。

**17.2.3.2 开始重新同步**

DRBD重新同步启动后，在本地生成标识符执行这些修改：

**图17.4 开始重新同步的GI元组的变化**

1.     当前同步源的UUID保持不变。

2.     同步源位图的UUID是旋转的第一个历史UUID。

3.     新的Bitmap（位图） UUID生成同步源。

4.     此UUID将成为新的当前同步目标的UUID。

5.     位图和历史的UUID同步目标保持不变。

**17.2.3.3 完成重新同步**

重新同步结束时，进行以下更改：

**图17.5 重新同步完成GI元组的变化**

1.     当前同步源的UUID保持不变。

2.     在同步源上的位图UUID翻转为第一个历史UUID，那个uuid移动为第二历史项（任何现有的第二个历史条项将被丢弃）。

3.     UUID同步源位图，然后清空（归零）。

4.     同步的目标采用同步源的整个GI tuple元组。

**17.2.4 如何DRBD使用代标识符**

当节点间建立连接，两节点就交换目前可用的代标识，并进行相应的处理。存在一些可能的如下结果：

**在两个节点上的当前UUID为空。**本地节点检测到其当前的UUID和对等节点当前的UUID是空的。这是一个新配置的资源，没有进行完整初始同步。没有同步发生，必须手动启动。

**目前的UUID在一个节点上空。**本地节点的检测对等节点的当前UUID为空，而它自己则不是。这是一个新配置的资源，刚启动初始完全同步的正常情况，本地节点被选中作为初始同步源。 DRBD在磁盘上的同步位图（这意味着它进行整个设备同步）现在设置的所有位，并开始作为一个同步源进行同步。 （当地当前UUID为空，对等节点非空），相反情况下，DRBD的执行相同的步骤，除非本地节点成为同步的目标。

**当前UUID相等。**本地节点检测到其当前的UUID和同行的当前的UUID非空且相等。当它处于次角色时发生了断开的情况，两个断开的节点都没有提升资源。没有同步发生，因次是没有必要。

**位图的UUID与对等节点的当前的UUID匹配**。本地节点检测到它的位图的UUID匹配对等节点当前的UUID，同时对等节点的的位图的UUID为空。这是当本地节点为主角色次要节点故障后的正常和预期发生的。这意味着在此期间对等节点从来没有成为主角色并在相同的数据生成的基础上都单独工作。现在DRBD正常启动，本地成为同步源节点，后台重新同步。反过来说，如果本地节点检测到它的位图的UUID是空的，同时对等节点的位图与本地节点的当前的UUID匹配，那么本地节点的失败后这就是正常的和预期发生。DRBD现在启动正常当地的节点成为同步目标，后台重新同步。

**当前的UUID匹配对等节点的历史UUID。**本地节点的检测当前的UUID匹配对等节点的的历史UUID之一。这意味着，两个数据集同父，而且本地节点是最新的数据，保存在本地节点的位图的信息是过时的和不可用的。因此，一个正常的同步是不够的。本地节点成为同步源，DRBD现在标识整个设备过时并初始化重新进行同步。在相反的情况下（本地节点的历史的UUID匹配对等节点当前的UUID），DRBD执行相同的步骤，除非本地节点成为同步的目标。

**位图的UUID匹配，当前UUID不匹配。**本地节点检测到当前的UUID和对等节点当前的UUID不同，而位图的UUID的匹配。这就是脑裂，但一个数据代具有相同的父。这意味着如果配置，DRBD的调用裂脑自动恢复策略。否则，DRBD断开连接，并等待人工脑裂进行处理。

**当前和位图UUID都不匹配。**本地节点检测到其当前的UUID和对等节点的UUID不同，同时位图的UUID也不匹配。这是脑裂且一个数据代具有不同的父，从而即使配置自动恢复策略，也没有实际意义。 DRBD断开，并等待人工脑裂进行处理。

**没有相匹配的UUID。**最后，DRBD技术的情况下无法检测到两个节点的GI tuples元组一个都不匹配，它记录一个警告一无关的数据并断开。这是DRBD意外连接两个节点的集群，彼此之间并不认识。

**17.3 活动日志**

**17.3.1 目的**

Drbd本地写操作支持写在Drbd本地块设备，同时也将写操作通过网络发送数据块。对所有的实际目的这两个动作都是同时发生。随机时序可能会导致一个写操作已经完成，但是通过网络的传输却未发生。

如果此时活动的节点故障，故障转移启动，则该数据块在两节点间不同步—它已在崩溃前写入失败的节点，但是尚未完成复制。因此，最终恢复节点时此块必须在随后的同步数据中删除。否则，失败的节点将是“一个超前写”的幸存节点，这将违反“全有全无”的存储原则。其实这不仅限于drbd，这个问题是在几乎所有复制的存储配置中都存在的问题。许多其他的存储解决方案（就像drbd本身，在0.7之前的版本），因此需呀i在活动的节点故障后进行完全同步，然后再修复。

在0.7版本之前的drbd是与之不同的。活动日志（AL）存储在元数据区，保持“最近”被写入块轨道。也就是俗称的热程度。

如果暂时失败的节点处于主动模式发生同步故障，只有那些在AL突出的热程度，而不是完整的设备需要同步。这大大减少了主动节点崩溃后的同步时间，。

**17.3.2 活跃程度**

活动日志有配置参数：活跃度。每个活跃度增加4MiB数据量用于主崩溃后的传输。这个参数必须被理解为以下方面之间的折中：

**大量的活跃度。**为调高读写吞吐量，保持一个大型的活动日志。每一个新活跃度被激活时，旧的不活跃的活跃度被复位。这种转变需要元数据区的写操作。如果活跃度高，旧的活跃度的换出则比较少见，减少元数据的写操作可以提高性能。

**很少的活跃度。**为减少同步时间则保持比较小的活动日志，主节点故障并随之恢复。

**17.3.3 选择合适的活动日志的大小**

应根据给定同步速率所需的同步时间定义活跃度的值。活跃度可利用如下公式计算：

**图17.6 基于同步率和目标同步时间的活跃程度计算**

R为同步速率，单位为MB/s，Tsync为目标同步时间，单位为秒（s）。E则为活跃度。

提供这样一个例子，假设集群有一个I/O子系统吞吐率90MiByte/s，而配置速率为30MiBytes/s（R=30），而同时想保持目标同步时间在4分钟或240s（Tsync=240）

**图17.7 活跃度的计算基于同步率和目标同步时间（例如）**

精确的计算结果为1800，但为保持DRBD hash功能的AL在最好的程度需设置为素数，因此选择1801。

**17.4 快速同步位图**

DRBD使用的内部数据结构为快速同步位图，基于每一个资源之上保持块的同步（在两个节点上相同）或者不同步。在节点处于断开模式时它是唯一相关的资源。

在快速同步位图中，一个位代表磁盘上一个4KIB块数据，如果该位被清零，就意味着对等节点的相应的块仍然是同步的。这就意味着从断开以来没有被写入的时间。相反，如果该位置被设置，就意味该块被修改，需要在连接重新可用时进行重新同步。

Drbd在断开连接的设备上检测写I/O，并因此在RAM中开始设置快速位图同步的位—这样避免了开销较大的同步元数据的I/O操作。只有相应的块变冷（即“活动日志”届满），drbd则对快速同步位图的位进行适当的修改。同样，如果资源被手动关闭，其余接上断开，则drbd刷新完整的快速同步位图为永久存储。

当对等节点恢复或重新建立连接，drbd结合两个节点的位图信息，确定必须重新同步的数据集。同时drbd检查生成的标示符以确定同步的方向。

作为同步源的节点传输商定的块到对等节点，清除同步目标确认的修改的位图中同步位。如果重新同步中断（如另一个网络中断），随后将从它离开的地方进行恢复—--当然，与此同时任何额外块的修改被添加到重新同步数据集。

|  |
| --- |
| 备注  重新同步也可以使用drbdadm pause-sync和drbdadm resume-sync命令进行暂停和恢复。然后操作并不轻松---中断重新同步会导致次节点的磁盘超过实际需要的不一致。 |

**17.5 对等节点的fencing接口**

Drbd有一个明确定义的接口，fences在对等节点以防止复制连接时被中断。drbd-peer-outdater帮助，参考绑定heartbeat以实现该接口。然而，可很容易的实现对等节点fencing的辅助程序。

Fencing帮助在下列情况相爱调用：

1.     fence-peer handler被定义为资源（或者common）处理的一部分。

2.     资源的fencing选项设置为resource-only 或者是 resource-and-stonith 。

3.     复制连接中断的时间对Drbd检测网络故障足够长。

当程序或者脚本作为fence-peer handler调用时，需要设置DRBD\_RESOURCE和DRBD\_PEER环境变量可用。它们分别包含受影响drbd资源的名字和对等节点的主机名。

任何对等节点的fencing帮助程序（或脚本）都必须返回以下退出的代码之一：

**表17.1 fence-peer handler退出代码**

| **退出代码** | **含义** |
| --- | --- |
| 3 | 对等节点的磁盘状态不一致（Inconsistent） |
| 4 | 对等节点磁盘状态成功设置为Outdated（或者是Outdated开始） |
| 5 | 对等节点连接失败，对等节点不可达。 |
| 6 | 因为受影响的资源为主角色拒绝了对等节点变为outdated |
| 7 | 对等节点集群成功fenced off。除非fencing设置受影响的资源为resource-and-stonith，否则不可能发生。 |

**18 获取更多信息**

目录

**18.1 商用DRBD支持**

drbd项目的赞助公司linbit提供对商用drbd支持、咨询和培训服务。

**18.2 公开邮件列表**

普通的使用问题可以反馈给公开邮箱：[drbd-user@lists.linbit.com](mailto:drbd-user@lists.linbit.com)

可通过<http://lists.linbit.com/drbd-user>订阅邮件。

完整的列表文档：<http://lists.linbit.com/pipermail/drbd-user>。

**18.3 公共IRC渠道**

一些Drbd开发有时可从公开的IRC服务irc.freenode.net，特别是一下渠道获得信息：

·         #drbd,

·         #linux-ha,

·         #linux-cluster.

在IRC上可以公开讨论给drbd的开发层面提建议。

**18.4 博客**

本指南的共同作者之一Florian Haas的技术的博客

Martin Loschwitz a.k.a. madkiss，另一个LINBIT雇员和合作的Debian DRBD技术包的维护者，在德国保持一个个人博客。

Planet HA是一个聚合信息的高可用性开发，技术顾问和用户的数量集中的博客。

**18.5 官方Twitter账户**

LINBIT有一个官方Twitter帐户：linbit。

如果呼叫DRBD，请注明#drbd hashtag。

**18.6 刊物**

Drbd的作者撰写和发表关于drbd或者drbd的某一方面的论文。下面为一个简单的列表：

Lars Ellenberg. *DRBD v8.0.x and beyond*. 2007. Available at<http://www.drbd.org/fileadmin/drbd/publications/drbd8.linux-conf.eu.2007.pdf>

Philipp Reisner. *DRBD v8 - Replicated Storage with Shared Disk Semantics*. 2007. Available at<http://www.drbd.org/fileadmin/drbd/publications/drbd8.pdf>.

Philipp Reisner. *Rapid resynchronization for replicated storage*. 2006. Available at<http://www.drbd.org/fileadmin/drbd/publications/drbd-activity-logging_v6.pdf>

**18.7 其他有用的资源**

维基百科有DRBD技术的项目http://en.wikipedia.org/wiki/DRBD。

无论是Linux - HA的wiki <http://wiki.linux-ha.org/> 还是ClusterLabs<http://www.clusterlabs.org/>都有一些利用DRBD技术在高可用性集群有用的信息。

**第七篇 附录**

http://www.drbd.org/users-guide-emb/p-appendices.html